Sybase® Adaptive Server™ Enterprise Reference Manual

Volume 1: Building Blocks

Adaptive Server Enterprise Version 12
Document ID: 36271-01-1200-02
Last Revised: October 1999

Principal author: Enterprise Data Studios Publications

Contributing authors: Anneli Meyer, Evelyn Wheeler

Document ID: 36271-01-1200

This publication pertains to Adaptive Server Enterprise Version 12 of the Sybase database management software and to any subsequent version until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Sybase, the SYBASE logo, Adaptive Server, APT-FORMS, Certified SYBASE Professional, the Certified SYBASE Professional logo, Column Design, ComponentPack, Data Workbench, First Impression, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, Transact-SQL, Visual Components, VisualWriter, and VQL are registered trademarks of Sybase, Inc.

Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server Enterprise Monitor, Adaptive Warehouse, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, EnterpriseConnect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript,

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

About This Book

1. System and User-Defined Datatypes

2. Transact-SQL Functions

3. Expressions, Identifiers, and Wildcard Characters

4. Reserved Words

5. SQLSTATE Codes and Messages

For the Index, see volume 4, "Tables and Reference Manual Index."

List of Tables

About This Book

The *Adaptive Server Reference Manual* is a four-volume guide to Sybase® Adaptive Server™ Enterprise and the Transact-SQL® language.

Volume 1, "*Building Blocks*," describes the "parts" of Transact-SQL: datatypes, built-in functions, expressions and identifiers, reserved words, and SQLSTATE errors. Before you can use Transact-SQL sucessfully, you need to understand the what these building blocks do and how they affect the results of Transact-SQL statements.

Volume 2, "*Commands*," provides reference information about the Transact-SQL commands, which you use to create statements.

Volume 3, "*Procedures*" provides reference information about system procedures, catalog stored procedures, extended stored procedures, and **dbcc** stored procedures. All procedures are created using Transact-SQL statements.

Volume 4, "*Tables and Reference Manual Index*," provides reference information about the system tables, which store information about your server, databases, users, and other information. It provides information about the tables in the *dbccdb* and *dbccalt* databases. It also contains an index that covers the topics of all four volumes.

Audience

The *Adaptive Server Reference Manual* is intended as a reference tool for Transact-SQL users of all levels.

How to Use This Book

This manual contains:

- Chapter 1, "System and User-Defined Datatypes," which describes the system and user-defined datatypes that are supplied with Adaptive Server and indicates how to use them to create user-defined datatypes.
- Chapter 2, "Transact-SQL Functions," which provides reference information for the Adaptive Server aggregate functions, datatype conversion functions, date functions, mathematical functions, row aggregate functions, string functions, system functions, and text and image functions.

- Chapter 3, “Expressions, Identifiers, and Wildcard Characters” which provides information about using the Transact-SQL language.
- Chapter 4, “Reserved Words,” which provides information about the Transact-SQL and SQL92 keywords.
- Chapter 5, “SQLSTATE Codes and Messages,” which contains information about Adaptive Server’s SQLSTATE status codes and the associated messages.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- The *Release Bulletin* for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the *Release Bulletin* may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use SyBooks™-on-the-Web.
- The Adaptive Server installation documentation for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *What’s New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12, the system changes added to support those features, and the changes that may affect your existing applications.
- *Transact-SQL User’s Guide* – documents Transact-SQL, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the *pubs2* and *pubs3* sample databases.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.
- *Adaptive Server Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.

- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Programs* manual for your platform – documents the Adaptive Server utility programs, such as **isql** and **bcp**, which are executed at the operating system level.
- *Error Messages* and *Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as datatypes and user-defined functions in the Adaptive Server database.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM Features in distributed transaction processing environments.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* provides instructions for using Sybase's DTM XA Interface with X/Open XA transaction managers.
- *Adaptive Server Glossary* – defines technical terms used in the Adaptive Server documentation.

Other Sources of Information

Use the Sybase Technical Library CD and the Technical Library Web site to learn more about your product:

- Technical Library CD contains product manuals and technical documents and is included with your software. The DynaText browser (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting Technical Library.

- Technical Library Web site includes the Product Manuals site, which is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition, you'll find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase / Powersoft newsgroups.

  To access the Technical Library Web site, go to support.sybase.com, click the Electronic Support Services tab, and select a link under the Technical Library heading.

Conventions Used in This Manual

The following sections describe conventions used in this manual.

Formatting SQL Statements

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented.

Font and Syntax Conventions

Table 1 shows the conventions for syntax statements that appear in this manual:

Table 1: Font and syntax conventions for this manual

Element	Example
Command names, command option names, utility names, utility options, and other keywords are bold.	**select** **sp_configure**
Database names, datatypes, file names and path names are in italics.	*master* database

Table 1: Font and syntax conventions for this manual (continued)

Element	Example
Variables, or words that stand for values that you fill in, are in italics.	`select` *column_name* `from` *table_name* `where` *search_conditions*
Type parentheses as part of the command.	`compute` *row_aggregate* `(`*column_name*`)`
Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.	`[cash | check | credit]`
The comma means you may choose as many of the options shown as you want. Separate your choices with commas as part of the command.	`cash, check, credit`
The pipe or vertical bar(|) means you may select only one of the options shown.	`cash | check | credit`

An ellipsis (...) means that you can **repeat** the last unit as many times as you like.	`buy thing = price [cash | check | credit]`
	`[, thing = price [cash | check | credit]]...`
	You must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

```
sp_dropdevice [device_name]
```

  or, for a command with more options:

```
select column_name
       from table_name
       where search_conditions
```

  In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

```
select * from publishers
```

- Examples of output from the computer appear as follows:

```
pub_id  pub_name                city         state
------- ------------------      ----------- -----
0736    New Age Books           Boston      MA
0877    Binnet & Hardley        Washington  DC
1389    Algodata Infosystems    Berkeley    CA

(3 rows affected)
```

Case

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, **SELECT**, **Select**, and **select** are the same.

Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see "Changing the Default Character Set, Sort Order, or Language" in Chapter 19, "Configuring Character Sets, Sort Orders, and Languages" in the *System Administration Guide*.

Expressions

Adaptive Server syntax statements use several different types of expressions.

Table 2: Types of expressions used in syntax statements

Usage	Definition
expression	Can include constants, literals, functions, column identifiers, variables or parameters
logical expression	An expression that returns TRUE, FALSE or UNKNOWN
constant expression	An expression that always returns the same value, such as "5+3" or "ABCDE"
float_expr	Any floating-point expression or expression that implicitly converts to a floating value
integer_expr	Any integer expression, or an expression that implicitly converts to an integer value

Table 2: Types of expressions used in syntax statements (continued)

Usage	Definition
numeric_expr	Any numeric expression that returns a single value
char_expr	Any expression that returns a single character-type value
binary_expression	An expression that returns a single *binary* or *varbinary* value

If You Need Help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

1 System and User-Defined Datatypes

This chapter describes the Transact-SQL datatypes. Datatypes specify the type, size, and storage format of columns, stored procedure parameters, and local variables. Topics covered are:

Datatype Categories

Adaptive Server provides several system datatypes and the user-defined datatypes *timestamp* and *sysname*. Table 1-1 lists the

categories of Adaptive Server datatypes. Each category is described in a section of this chapter.

Table 1-1: Datatype categories

Category	Used For
Exact Numeric Datatypes	Numeric values (both integers and numbers with a decimal portion) that must be represented exactly
Approximate Numeric Datatypes	Numeric data that can tolerate rounding during arithmetic operations
Money Datatypes	Monetary data
Timestamp Datatype	Tables that are browsed in Client-Library™ applications
Date and Time Datatypes	Date and time information
Character Datatypes	Strings consisting of letters, numbers, and symbols
Binary Datatypes	Raw binary data, such as pictures, in a hexadecimal-like notation
bit Datatypes	True/false and yes/no type data
sysname Datatype	System tables
text and image Datatypes	Printable characters or hexadecimal-like data that requires more than 255 bytes of storage
User-Defined Datatypes	Defining objects that inherit the rules, default, null type, IDENTITY property, and base datatype

Range and Storage Size

Table 1-2 lists the system-supplied datatypes and their synonyms and provides information about the range of valid values and storage size for each. For simplicity, the datatypes are printed in lowercase characters, although Adaptive Server allows you to use either uppercase or lowercase characters for system datatypes. User-defined datatypes, such as *timestamp*, are **case sensitive**. Most

Adaptive Server-supplied datatypes are not reserved words and can be used to name other objects.

Table 1-2: Range and storage size for system datatypes

Datatypes	Synonyms	Range	Bytes of Storage
Exact numeric datatypes			
tinyint		0 to 255	1
smallint		-2^{15} (-32,768) to 215^{-1} (32,767)	2
int	*integer*	-2^{31} (-2,147,483,648) to 2^{31} -1 (2,147,483,647)	4
numeric (p, s)		-10^{38} to 10^{38} -1	2 to 17
decimal (p, s)	*dec*	-10^{38} to 10^{38} -1	2 to 17
Approximate numeric datatypes			
float (precision)		Machine dependent	4 or 8
double precision		Machine dependent	8
real		Machine dependent	4
Money datatypes			
smallmoney		-214,748.3648 to 214,748.3647	4
money		-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
Date/time datatypes			
smalldatetime		January 1, 1900 to June 6, 2079	4
datetime		January 1, 1753 to December 31, 9999	8
Character datatypes			
char(n)	*character*	255 characters or fewer	*n*
varchar(n)	*char[acter] varying*	255 characters or fewer	actual entry length
nchar(n)	*national char[acter]*	255 characters or fewer	*n* * *@@ncharsize*
nvarchar(n)	*nchar varying, national char[acter] varying*	255 characters or fewer	*n*

Table 1-2: Range and storage size for system datatypes (continued)

Datatypes	Synonyms	Range	Bytes of Storage
Binary datatypes			
binary(n)		255 bytes or fewer	*n*
varbinary(n)		255 bytes or fewer	actual entry length
Bit datatype			
bit		0 or 1	1 (1 byte holds up to 8 *bit* columns)
Text and image datatypes			
text		2^{31} -1 (2,147,483,647) bytes or fewer	0 until initialized, then a multiple of 2K
image		2^{31} -1 (2,147,483,647) bytes or fewer	0 until initialized, then a multiple of 2K

Declaring the Datatype of a Column, Variable, or Parameter

You must declare the datatype for a column, local variable, or parameter. The datatype can be any of the system-supplied datatypes or any user-defined datatype in the database.

Declaring the Datatype for a Column in a Table

Use the following syntax to declare the datatype of a new column in a **create table** or an **alter table** statement:

```
create table [[database.]owner.]table_name
   (column_name datatype [identity | not null | null]
       [, column_name datatype [identity | not null |
           null]]...)

alter table [[database.]owner.]table_name
   add column_name datatype [identity | null
       [, column_name datatype [identity | null]...
```

For example:

```
create table sales_daily
    (stor_id char(4)not null,
     ord_num numeric(10,0)identity,
     ord_amt money null)
```

Declaring the Datatype for a Local Variable in a Batch or Procedure

Use the following syntax to declare the datatype for a local variable in a batch or stored procedure:

```
declare @variable_name datatype
   [, @variable_name datatype]...
```

For example:

```
declare @hope money
```

Declaring the Datatype for a Parameter in a Stored Procedure

Use the following syntax to declare the datatype for a parameter in a stored procedure:

```
create procedure [owner.]procedure_name [;number]
   [[(]@parameter_name datatype [= default] [output]
       [,@parameter_name datatype [= default]
           [output]]...[)]]
   [with recompile]
   as SQL_statements
```

For example:

```
create procedure auname_sp @auname varchar(40)
as
    select au_lname, title, au_ord
    from authors, titles, titleauthor
    where @auname = au_lname
    and authors.au_id = titleauthor.au_id
    and titles.title_id = titleauthor.title_id
```

Determining the Datatype of a Literal

You cannot declare the datatype of a literal. Adaptive Server treats all character literals as *varchar*. Numeric literals entered with E notation are treated as *float*; all others are treated as exact numerics:

- Literals between $2^{31} - 1$ and -2^{31} with no decimal point are treated as *integer*.
- Literals that include a decimal point, or that fall outside the range for integers, are treated as *numeric*.

➤ ***Note***

To preserve backward compatibility, use E notation for numeric literals that should be treated as *float*.

Datatype of Mixed-Mode Expressions

When you perform concatenation or mixed-mode arithmetic on values with different datatypes, Adaptive Server must determine the datatype, length, and precision of the result.

Determining the Datatype Hierarchy

Each system datatype has a **datatype hierarchy**, which is stored in the *systypes* system table. User-defined datatypes inherit the hierarchy of the system datatype on which they are based.

The following query ranks the datatypes in a database by hierarchy. In addition to the information shown below, your query results will include information about any user-defined datatypes in the database:

```
select name,hierarchy
from systypes
order by hierarchy
```

```
name                          hierarchy
----------------------------  ---------
floatn                               1
float                                2
datetimn                             3
datetime                             4
real                                 5
numericn                             6
numeric                              7
decimaln                             8
decimal                              9
moneyn                              10
money                               11
smallmoney                          12
smalldatetime                       13
intn                                14
int                                 15
smallint                            16
tinyint                             17
```

```
bit                                  18
varchar                              19
sysname                              19
nvarchar                             19
char                                 20
nchar                                20
varbinary                            21
timestamp                            21
binary                               22
text                                 23
image                                24
(28 rows affected)
```

The datatype hierarchy determines the results of computations using values of different datatypes. The result value is assigned the datatype that is closest to the top of the list.

In the following example, *qty* from the *sales* table is multiplied by *royalty* from the *roysched* table. *qty* is a *smallint*, which has a hierarchy of 16; *royalty* is an *int*, which has a hierarchy of 15. Therefore, the datatype of the result is an *int*.

smallint(qty) * int(royalty) = int

Determining Precision and Scale

For *numeric* and *decimal* datatypes, each combination of precision and scale is a distinct Adaptive Server datatype. If you perform arithmetic on two *numeric* or *decimal* values:

- *n1* with precision *p1* and scale *s1*, and
- *n2* with precision *p2* and scale *n2*

Adaptive Server determines the precision and scale of the results as shown in Table 1-3:

Table 1-3: Precision and scale after arithmetic operations

Operation	Precision	Scale
n1 + n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 - n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 * n2	s1 + s2 + (p1 - s1) + (p2 - s2) + 1	s1 + s2
n1 / n2	max(s1 + p2 + 1, 6) + p1 - s1 + p2	max(s1 + p2 -s2 + 1, 6)

Converting One Datatype to Another

Many conversions from one datatype to another are handled automatically by Adaptive Server. These are called implicit conversions. Other conversions must be performed explicitly with the **convert**, **inttohex**, and **hextoint** functions. See "Datatype Conversion Functions" in Chapter 2, "Transact-SQL Functions," for details about datatype conversions supported by Adaptive Server.

Automatic Conversion of Fixed-Length NULL Columns

Only columns with variable-length datatypes can store null values. When you create a NULL column with a fixed-length datatype, Adaptive Server automatically converts it to the corresponding variable-length datatype. Adaptive Server does not inform the user of the datatype change.

Table 1-4 lists the fixed- and variable-length datatypes to which they are converted. Certain variable-length datatypes, such as *moneyn*, are reserved datatypes; you cannot use them to create columns, variables, or parameters:

Table 1-4: Automatic conversion of fixed-length datatypes

Original Fixed-Length Datatype	Converted To
char	*varchar*
nchar	*nvarchar*
binary	*varbinary*
datetime	*datetimn*
float	*floatn*
int, *smallint*, and *tinyint*	*intn*
decimal	*decimaln*
numeric	*numericn*
money and *smallmoney*	*moneyn*

Handling Overflow and Truncation Errors

The **arithabort** option determines how Adaptive Server behaves when an arithmetic error occurs. The two **arithabort** options, **arithabort arith_overflow** and **arithabort numeric_truncation**, handle different types of arithmetic errors. You can set each option independently, or set both options with a single **set arithabort on** or **set arithabort off** statement.

- **arithabort arith_overflow** specifies behavior following a divide-by-zero error or a loss of precision during either an explicit or an implicit datatype conversion. This type of error is considered serious. The default setting, **arithabort arith_overflow on**, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, **arithabort arith_overflow on** does not roll back earlier commands in the batch, but Adaptive Server does not execute any statements that follow the error-generating statement in the batch.

  If you set **arithabort arith_overflow off**, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.

- **arithabort numeric_truncation** specifies behavior following a loss of scale by an exact numeric datatype during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, **arithabort numeric_truncation on**, aborts the statement that causes the error but continues to process other statements in the transaction or batch. If you set **arithabort numeric_truncation off**, Adaptive Server truncates the query results and continues processing.

The **arithignore** option determines whether Adaptive Server prints a warning message after an overflow error. By default, the **arithignore** option is turned **off.** This causes Adaptive Server to display a warning message after any query that results in numeric overflow. To ignore overflow errors, use **set arithignore on.**

➤ ***Note***

The **arithabort** and **arithignore** options were redefined for release 10.0. If you use these options in your applications, examine them to be sure they still produce the desired effects.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL provides the *smallint*, *int*, *numeric*, *decimal*, *float*, *double precision*, *real*, *char*, and *varchar* SQL92 datatypes. The *tinyint*, *binary*, *varbinary*, *image*, *bit*, *datetime*, *smalldatetime*, *money*, *smallmoney*, *nchar*, *nvarchar*, *sysname*, *text*, *timestamp*, and user-defined datatypes are Transact-SQL extensions.

Exact Numeric Datatypes

Function

Use the exact numeric datatypes when it is important to represent a value exactly. Adaptive Server provides exact numeric types for both integers (whole numbers) and numbers with a decimal portion.

Integer Types

Adaptive Server provides three exact numeric datatypes to store integers: *int* (or *integer*), *smallint*, and *tinyint*. Choose the integer type based on the expected size of the numbers to be stored. Internal storage size varies by type, as shown in Table 1-5:

Table 1-5: Integer datatypes

Datatype	Stores	Bytes of Storage
int[eger]	Whole numbers between -2^{31} and $2^{31} - 1$ (-2,147,483,648 and 2,147,483,647), inclusive.	4
smallint	Whole numbers between -2^{15} and $2^{15} - 1$ (-32,768 and 32,767), inclusive.	2
tinyint	Whole numbers between 0 and 255, inclusive. (Negative numbers are not permitted.)	1

Entering Integer Data

Enter integer data as a string of digits without commas. Integer data can include a decimal point as long as all digits to the right of the decimal point are zeros. The *smallint* and *integer* datatypes can be preceded by an optional plus or minus sign. The *tinyint* datatype can be preceded by an optional plus sign.

Table 1-6 shows some valid entries for a column with a datatype of *integer* and indicates how **isql** displays these values:

Table 1-6: Valid integer values

Value Entered	Value Displayed
2	2
+2	2
-2	-2
2.	2
2.000	2

Table 1-7 lists some invalid entries for an *integer* column:

Table 1-7: Invalid integer values

Value Entered	Type of Error
2,000	Commas not allowed.
2-	Minus sign should precede digits.
3.45	Digits to the right of the decimal point are nonzero digits.

Decimal Datatypes

Adaptive Server provides two other exact numeric datatypes, *numeric* and *dec*[*imal*], for numbers that include decimal points. Data stored in *numeric* and *decimal* columns is packed to conserve disk space, and preserves its accuracy to the least significant digit after arithmetic operations. The *numeric* and *decimal* datatypes are identical in all respects but one: only *numeric* datatypes with a scale of 0 can be used for the IDENTITY column.

Specifying Precision and Scale

The *numeric* and *decimal* datatypes accept two optional parameters, *precision* and *scale*, enclosed in parentheses and separated by a comma:

datatype **[(*precision* [, *scale*])]**

Adaptive Server treats each combination of precision and scale as a distinct datatype. For example, *numeric*(10,0) and *numeric*(5,0) are two separate datatypes. The *precision* and *scale* determine the range of values that can be stored in a decimal or numeric column:

- The precision specifies the maximum number of decimal digits that can be stored in the column. It includes **all** digits, both to the right and to the left of the decimal point. You can specify precisions ranging from 1 digit to 38 digits or use the default precision of 18 digits.
- The scale specifies the maximum number of digits that can be stored to the right of the decimal point. The scale must be less than or equal to the precision. You can specify a scale ranging from 0 digits to 38 digits or use the default scale of 0 digits.

Storage Size

The storage size for a *numeric* or *decimal* column depends on its precision. The minimum storage requirement is 2 bytes for a 1- or 2-digit column. Storage size increases by approximately 1 byte for each additional 2 digits of precision, up to a maximum of 17 bytes.

Use the following formula to calculate the exact storage size for a *numeric* or *decimal* column:

```
ceiling (precision / log 256 ) + 1
```

For example, the storage size for a *numeric*(18,4) column is 9 bytes.

Entering Decimal Data

Enter *decimal* and *numeric* data as a string of digits preceded by an optional plus or minus sign and including an optional decimal point. If the value exceeds either the precision or scale specified for the column, Adaptive Server returns an error message. Exact numeric types with a scale of 0 are displayed without a decimal point.

Table 1-8 shows some valid entries for a column with a datatype of *numeric*(5,3) and indicates how these values are displayed by **isql**:

Table 1-8: Valid decimal values

Value Entered	Value Displayed
12.345	12.345
+12.345	12.345
-12.345	-12.345
12.345000	12.345
12.1	12.100
12	12.000

Table 1-9 shows some invalid entries for a column with a datatype of *numeric*(5,3):

Table 1-9: Invalid decimal values

Value Entered	Type of Error
1,200	Commas not allowed.
12-	Minus sign should precede digits.

Table 1-9: Invalid decimal values (continued)

Value Entered	Type of Error
12.345678	Too many nonzero digits to the right of the decimal point.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL provides the *smallint*, *int*, *numeric*, and *decimal* SQL92 exact numeric datatypes. The *tinyint* type is a Transact-SQL extension.

Approximate Numeric Datatypes

Function

Use the approximate numeric types, *float*, *double precision*, and *real*, for numeric data that can tolerate rounding during arithmetic operations. The approximate numeric types are especially suited to data that covers a wide range of values. They support all aggregate functions and all arithmetic operations except **modulo**.

Understanding Approximate Numeric Datatypes

Approximate numeric datatypes, used to store floating-point numbers, are inherently slightly inaccurate in their representation of real numbers—hence the name "approximate numeric". In order to use these datatypes, you must understand and accept their limitations.

Any time a floating-point number is printed or displayed, the printed representation is not quite the same as the stored number, and the stored number is not quite the same as the number that the user entered. Most of the time, the stored representation is close enough, and software makes the printed output look just like the original input, but you must understand the inaccuracy if you plan to use floating-point numbers for calculations, particularly if you will be doing repeated calculations using approximate numeric datatypes—the results can be surprisingly and unexpectedly inaccurate.

The inaccuracy occurs because floating-point numbers are stored in the computer as binary fractions (that is, as a representative number divided by a power of 2), but the numbers we use are decimal (powers of 10). This means that only a very small set of numbers can be stored accurately: 0.75 (3 / 4) can be stored accurately because it is a binary fraction (4 is a power of 2); 0.2 (2 / 10) can not (10 is not a power of 2).

Some numbers contain too many digits to store accurately. *double precision* is stored as 8 binary bytes and can represent about 17 digits with reasonable accuracy. *real* is stored as 4 binary bytes and can represent only about 6 digits with reasonable accuracy.

If you begin with numbers that are almost correct, and do computations with them using other numbers that are almost correct, you can easily end up with a result that is not even close to being correct. If these considerations are important to your application, consider using an exact numeric datatype.

Range, Precision, and Storage Size

The *real* and *double precision* types are built on types supplied by the operating system. The *float* type accepts an optional binary precision in parentheses. *float* columns with a precision of 1–15 are stored as *real*; those with higher precision are stored as *double precision*.

The range and storage precision for all three types is machine dependent.

Table 1-10 shows the range and storage size for each approximate numeric type. Note that **isql** displays only 6 significant digits after the decimal point and rounds the remainder:

Table 1-10: Approximate numeric datatypes

Datatype	Bytes of Storage
float[(default precision)]	4 for *default precision* < 16 8 for *default precision* >= 16
double precision	8
real	4

Entering Approximate Numeric Data

Enter approximate numeric data as a mantissa followed by an optional exponent:

- The mantissa is a signed or unsigned number, with or without a decimal point. The column's binary precision determines the maximum number of binary digits allowed in the mantissa.
- The exponent, which begins with the character "e" or "E," must be a whole number.

The value represented by the entry is the following product:

$$mantissa * 10^{EXPONENT}$$

For example, 2.4E3 represents the value 2.4 times 10^3, or 2400.

Standards and Compliance

Standard	Compliance Level
SQL92	The *float*, *double precision*, and *real* datatypes are entry level compliant.

Money Datatypes

Function

Use the *money* and *smallmoney* datatypes to store monetary data. You can use these types for U.S. dollars and other decimal currencies, but Adaptive Server provides no means to convert from one currency to another. You can use all arithmetic operations except **modulo**, and all aggregate functions, with *money* and *smallmoney* data.

Accuracy

Both *money* and *smallmoney* are accurate to one ten-thousandth of a monetary unit, but they round values up to two decimal places for display purposes. The default print format places a comma after every three digits.

Range and Storage Size

Table 1-11 summarizes the range and storage requirements for money datatypes:

Table 1-11: Money datatypes

Datatype	Range	Bytes of Storage
money	Monetary values between +922,337,203,685,477.5807 and -922,337,203,685,477.5808	8
smallmoney	Monetary values between +214,748.3647 and -214,748.3648	4

Entering Monetary Values

Monetary values entered with E notation are interpreted as *float*. This may cause an entry to be rejected or to lose some of its precision when it is stored as a *money* or *smallmoney* value.

money and *smallmoney* values can be entered with or without a preceding currency symbol, such as the dollar sign ($), yen sign (¥), or pound sterling sign (£). To enter a negative value, place the minus sign after the currency symbol. Do not include commas in your entry.

Standards and Compliance

Standard	Compliance Level
SQL92	The *money* and *smallmoney* datatypes are Transact-SQL extensions.

Timestamp Datatype

Function

Use the user-defined *timestamp* datatype in tables that are to be browsed in Client-Library™ applications (see "Browse Mode" for more information). Adaptive Server updates the *timestamp* column each time its row is modified. A table can have only one column of *timestamp* datatype.

Datatype Definition

timestamp is an Adaptive Server-supplied, user-defined datatype that is defined as *varbinary(8)* NULL. It requires 8 bytes of storage. Because *timestamp* is a user-defined datatype, you cannot use it to define other user-defined datatypes. You cannot use the aggregate functions **sum** or **avg** with the *timestamp* datatype.

Unlike the SQL standard *timestamp* datatype, the Transact-SQL *timestamp* datatype does not hold date and time information, and cannot be converted to a date and time. *timestamp* holds binary-type data like that shown below:

```
 timestamp
------------------
 0x0001000000000e51
```

Creating a *timestamp* Column

If you create a column named *timestamp* without specifying a datatype, Adaptive Server defines the column as a *timestamp* datatype:

```
create table testing
    (c1 int, timestamp, c2 int)
```

You can also explicitly assign the *timestamp* datatype to a column named *timestamp*:

```
create table testing
    (c1 int, timestamp timestamp, c2 int)
```

or to a column with another name:

```
create table testing
    (c1 int, t_stamp timestamp,c2 int)
```

You can create a column named *timestamp* and assign it another datatype (although this could be confusing to other users and would not allow the use of the **browse** functions in Open Client™ or with the **tsequal** function):

```
create table testing
    (c1 int, timestamp datetime)
```

Standards and Compliance

Standard	Compliance Level
SQL92	The *timestamp* datatype is a Transact-SQL extension.

Date and Time Datatypes

Function

Use the *datetime* and *smalldatetime* datatypes to store absolute date and time information.

Adaptive Server also provides, which store binary-type information.

Range and Storage Requirements

Table 1-12 summarizes the range and storage requirements for the *datetime* and *smalldatetime* datatypes:

Table 1-12: Transact-SQL datatypes for storing dates and times

Datatype	Range	Bytes of Storage
datetime	January 1, 1753 through December 31, 9999	8
smalldatetime	January 1, 1900 through June 6, 2079	4

Entering *datetime* and *smalldatetime* Data

The *datetime* and *smalldatetime* datatypes consist of a date portion either followed by or preceded by a time portion. (You can omit either the date or the time, or both.) Both *datetime* and *smalldatetime* values must be enclosed in single or double quotes.

- *datetime* columns hold dates between January 1, 1753 and December 31, 9999. *datetime* values are accurate to 1/300 of a second on platforms that support this level of granularity. Storage size is 8 bytes: 4 bytes for the number of days since the base date of January 1, 1900 and 4 bytes for the time of day.
- *smalldatetime* columns hold dates from January 1, 1900 to June 6, 2079, with accuracy to the minute. Storage size is 4 bytes: 2 bytes for the number of days since January 1, 1900 and 2 bytes for the number of minutes since midnight.

Entering the Date Portion of a *datetime* or *smalldatetime* Value

Dates consist of a month, day, and year and can be entered in a variety of formats:

- You can enter the entire date as an unseparated string of 4, 6, or 8 digits, or use slash(/), hyphen (-), or period(.) separators between the date parts.

  - When entering dates as unseparated strings, use the appropriate format for that string length. Use leading zeros for single-digit years, months, and days. Dates entered in the wrong format may be misinterpreted or result in errors.
  - When entering dates with separators, use the **set dateformat** option to determine the expected order of date parts. If the first date part in a separated string is four digits, Adaptive Server interprets the string as *yyyy-mm-dd* format.

- Some date formats accept 2-digit years (*yy*). Dates greater than or equal to 50 are interpreted as 19yy; those less than 50 are interpreted as 20yy.
- You can specify the month as either a number or a name. Month names and their abbreviations are language-specific and can be entered in uppercase, lowercase, or mixed case.
- If you omit the date portion of a *datetime* or *smalldatetime* value, Adaptive Server uses the default date of January 1, 1900.

Table 1-13 describes the acceptable formats for entering the date portion of a *datetime* or *smalldatetime* value:

Table 1-13: Date formats for datetime and smalldatetime datatypes

Date Format	Interpretation	Sample Entries	Meaning
4-digit string with no separators	Interpreted as *yyyy*. Date defaults to Jan 1 of the specified year.	"1947"	Jan 1 1947
6-digit string with no separators	Interpreted as *yymmdd*. For *yy* < 50, year is 20*yy*. For *yy* >= 50, year is 19*yy*.	"450128" "520128"	Jan 28 2045 Jan 28 1952
8-digit string with no separators	Interpreted as *yyyymmdd*.	"19940415"	Apr 15 1994
String consisting of 2-digit month, day, and year separated by slashes, hyphens, or periods, or a combination of the above.	The **dateformat** and **language set** options determine the expected order of date parts. For us_english, the default order is *mdy*. For *yy* < 50, year is interpreted as 20*yy*. For *yy* >= 50, year is interpreted as 19*yy*.	"4/15/94" "4.15.94" "4-15-94" "04.15/94"	All of these entries are interpreted as Apr 15 1994 when the **dateformat** option is set to **mdy**.

Table 1-13: Date formats for datetime and smalldatetime datatypes (continued)

Date Format	Interpretation	Sample Entries	Meaning
String consisting of 2-digit month, 2-digit day, and 4-digit year separated by slashes, hyphens, or periods, or a combination of the above.	The **dateformat** and **language set** options determine the expected order of date parts. For us_english, the default order is *mdy*.	"04/15.1994"	Interpreted as Apr 15 1994 when the **dateformat** option is set to **mdy**.
Month is entered in character form (either full month name or its standard abbreviation), followed by an optional comma.	If 4-digit year is entered, date parts can be entered in any order.	"April 15, 1994" "1994 15 apr" "1994 April 15" "15 APR 1994"	All of these entries are interpreted as Apr 15 1994.
	If day is omitted, all 4 digits of year must be specified. Day defaults to the first day of the month.	"apr 1994"	Apr 1 1994
	If year is only 2 digits (*yy*), it is expected to appear after the day. For *yy* < 50, year is interpreted as 20*yy*. For *yy* >= 50, year is interpreted as 19*yy*.	"mar 16 17" "apr 15 94"	Mar 16 2017 Apr 15 1994
The empty string, ""	Date defaults to Jan 1 1900.	""	Jan 1 1900

Entering the Time Portion of a *datetime* or *smalldatetime* Value

The time component of a *datetime* or *smalldatetime* value must be specified as follows:

***hours*[:*minutes*[:*seconds*[:m*illiseconds*]] [AM | PM]**

- Use 12AM for midnight and 12PM for noon.
- A time value must contain either a colon or an AM or PM signifier. The AM or PM can be entered in uppercase, lowercase, or mixed case.
- The seconds specification can include either a decimal portion preceded by a decimal point or a number of milliseconds preceded by a colon. For example, "12:30:20:1" means twenty seconds and one millisecond past 12:30; "12:30:20.1" means twenty and one-tenth of a second past 12:30.
- If you omit the time portion of a *datetime* or *smalldatetime* value, Adaptive Server uses the default time of 12:00:00:000AM.

Display Formats for *datetime* and *smalldatetime* Values

The display format for *datetime* and *smalldatetime* values is "Mon dd yyyy hh:mmAM" (or "PM"); for example, "Apr 15 1988 10:23PM". To display seconds and milliseconds, and to obtain additional date styles and date-part orders, use the **convert** function to convert the data to a character string. Adaptive Server may round or truncate millisecond values.

Table 1-14 lists some examples of *datetime* entries and their display values:

Table 1-14: Examples of datetime entries

Entry	Value Displayed
"1947"	Jan 1 1947 12:00AM
"450128 12:30:1PM"	Jan 28 2045 12:30PM
"12:30.1PM 450128"	Jan 28 2045 12:30PM
"14:30.22"	Jan 1 1900 2:30PM
"4am"	Jan 1 1900 4:00AM

Finding *datetime* Values That Match a Pattern

Use the **like** keyword to look for dates that match a particular pattern. If you use the equality operator (=) to search *datetime* values for a particular month, day, and year, Adaptive Server returns only those values for which the time is precisely 12:00:00:000AM.

For example, if you insert the value "9:20" into a column named *arrival_time*, Adaptive Server converts the entry into "Jan 1 1900 9:20AM". If you look for this entry using the equality operator, it is not found:

```
where arrival_time = "9:20" /* does not match */
```

You can find the entry using the **like** operator:

```
where arrival_time like "%9:20%"
```

When using **like**, Adaptive Server first converts the dates to *datetime* format and then to *varchar*. The display format consists of the 3-character month in the current language, 2 characters for the day, 4 characters for the year, the time in hours and minutes, and "AM" or "PM."

When searching with **like**, you cannot use the wide variety of input formats that are available for entering the date portion of *datetime*

and *smalldatetime* values. Since the standard display formats do not include seconds or milliseconds, you cannot search for seconds or milliseconds with **like** and a match pattern, unless you are also using *style* 9 or 109 and the **convert** function.

If you are using **like**, and the day of the month is a number between 1 and 9, insert 2 spaces between the month and the day to match the *varchar* conversion of the *datetime* value. Similarly, if the hour is less than 10, the conversion places 2 spaces between the year and the hour. The clause:

```
like May 2%
```

(with 1 space between "May" and "2") finds all dates from May 20 through May 29, but not May 2. You do not need to insert the extra space with other date comparisons, only with **like**, since the *datetime* values are converted to *varchar* only for the **like** comparison.

Manipulating Dates

You can do some arithmetic calculations on *datetime* values with the built-in date functions. See "Date Functions" in Chapter 2, "Transact-SQL Functions."

Standards and Compliance

Standard	Compliance Level
SQL92	The *datetime* and *smalldatetime* datatypes are Transact-SQL extensions.

Character Datatypes

Function

Use the character datatypes to store strings consisting of letters, numbers, and symbols. Use the fixed-length datatype, *char(n)*, and the variable-length datatype, *varchar(n)*, for single-byte character sets such as us_english. Use the fixed-length datatype, *nchar(n)*, and the variable-length datatype, *nvarchar(n)*, for multibyte character sets such as Japanese. The character datatypes can store a maximum of 255 characters; use the *text* datatype (described in "text and image Datatypes") for strings longer than 255 characters.

Length and Storage Size

Use *n* to specify the length in characters for the fixed-length datatypes, *char(n)* and *nchar(n)*. Entries shorter than the assigned length are blank-padded; entries longer than the assigned length are truncated without warning, unless the **string_rtruncation** option to the **set** command is set to **on**. Fixed-length columns that allow nulls are internally converted to variable-length columns.

Use *n* to specify the maximum length in characters for the variable-length datatypes, *varchar(n)* and *nvarchar(n)*. Data in variable-length columns is stripped of trailing blanks; storage size is the actual length of the data entered. Data in variable-length variables and parameters retains all trailing blanks, but is not padded to the defined length. Character literals are treated as variable-length datatypes.

Fixed-length columns tend to take more storage space than variable-length columns, but are accessed somewhat faster. Table 1-15 summarizes the storage requirements of the different character datatypes:

Table 1-15: Character datatypes

Datatype	Stores	Bytes of Storage
char(n)	Fixed-length data, such as social security numbers or postal codes, in single-byte character sets.	*n*
nchar(n)	Fixed-length data in multibyte character sets	*n* * *@@ncharsize*
varchar(n)	Variable-length data, such as names, in single-byte character sets.	Actual number of characters entered

Table 1-15: Character datatypes (continued)

Datatype	Stores	Bytes of Storage
nvarchar(n)	Variable-length data in multibyte character sets	Actual number of characters * *@@ncharsize*

Determining Column Length with System Functions

Use the **char_length** string function and **datalength** system function to determine column length:

- **char_length** returns the number of characters in the column, stripping trailing blanks for variable-length datatypes.
- **datalength** returns the number of bytes, stripping trailing blanks for data stored in variable-length columns.

When a *char* value is declared to allow NULLS, SQL Server stores it internally as a *varchar.*

Entering Character Data

Character strings must be enclosed in single or double quotes. If you use **set quoted_identifier on**, use single quotes for character strings; otherwise, Adaptive Server treats them as identifiers.

Strings that include the double-quote character should be surrounded by single quotes. Strings that include the single-quote character should be surrounded by double quotes. For example:

```
'George said, "There must be a better way."'
"Isn't there a better way?"
```

An alternative is to enter two quotation marks for each quotation mark you want to include in the string. For example:

```
"George said, ""There must be a better way.""
'Isn''t there a better way?'
```

To continue a character string onto the next line of your screen, enter a backslash (\) before going to the next line.

Treatment of Blanks

The following example creates a table named *spaces* that has both fixed- and variable-length character columns:

```
create table spaces (cnot char(5) not null,
    cnull char(5) null,
    vnot varchar(5) not null,
        vnull varchar(5) null,
    explanation varchar(25) not null)

insert spaces values ("a", "b", "c", "d",
    "pads char-not-null only")
insert spaces values ("1    ", "2    ", "3    ",
    "4    ", "truncates trailing blanks")
insert spaces values ("    e", "    f", "    g",
     "    h", "leading blanks, no change")
insert spaces values ("   w ", "   x ", "   y ",
    "   z ", "truncates trailing blanks")
insert spaces values ("", "", "", "",
    "empty string equals space" )

select "[" + cnot + "]",
       "[" + cnull + "]",
       "[" + vnot + "]",
       "[" + vnull + "]",
    explanation from spaces
```

```
                                 explanation
-------  -------  ------- ------- --------------------
[a    ]  [b]      [c]     [d]     pads char-not-null only
[1    ]  [2]      [3]     [4]     truncates trailing blanks
[    e]  [    f]  [    g] [    h] leading blanks, no change
[   w ]  [   x]   [   y]  [   z]  truncates trailing blanks
[     ]  [ ]      [ ]     [ ]     empty string equals space

(5 rows affected)
```

This example illustrates how the column's datatype and null type interact to determine how blank spaces are treated:

- Only *char* **not null** and *nchar* **not null** columns are padded to the full width of the column; *char* **null** columns are treated like *varchar* and *nchar* **null** columns are treated like *nvarchar.*
- Preceding blanks are not affected.
- Trailing blanks are truncated except for *char* and *nchar* **not null** columns.
- The empty string ("") is treated as a single space. In *char* and *nchar* **not null** columns, the result is a column-length field of spaces.

Manipulating Character Data

You can use the **like** keyword to search character strings for particular characters and the built-in string functions to manipulate their contents. Strings consisting of numbers can be used for arithmetic after being converted to exact and approximate numeric datatypes with the **convert** function.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL provides the *char* and *varchar* SQL92 datatypes. The *nchar* and *nvarchar* datatypes are Transact-SQL extensions.

Binary Datatypes

Function

Use the binary datatypes, *binary(n)* and *varbinary(n)*, to store up to 255 bytes of raw binary data, such as pictures, in a hexadecimal-like notation.

Valid Binary and Varbinary Entries

Binary data begins with the characters "0x" and can include any combination of digits and the uppercase and lowercase letters A through F.

Use *n* to specify the column length in bytes, or use the default length of 1 byte. Each byte stores 2 binary digits. If you enter a value longer than *n*, Adaptive Server truncates the entry to the specified length without warning or error.

Use the fixed-length binary type, *binary(n)*, for data in which all entries are expected to be approximately equal in length.

Use the variable-length binary type, *varbinary(n)*, for data that is expected to vary greatly in length.

Because entries in *binary* columns are zero-padded to the column length (*n*), they may require more storage space than those in *varbinary* columns, but they are accessed somewhat faster.

Use the *image* Datatype for Entries of More Than 255 Bytes

Use the *image* datatype to store larger blocks of binary data (up to 2,147,483,647 bytes) on external data pages. You cannot use the *image* datatype for variables or for parameters in stored procedures. For more information, see the section "text and image Datatypes."

Treatment of Trailing Zeros

All *binary* **not null** columns are padded with zeros to the full width of the column. Trailing zeros are truncated in all *varbinary* data and in *binary* **null** columns, since columns that accept null values must be treated as variable-length columns.

The following example creates a table with all four variations of *binary* and *varbinary* datatypes, NULL and NOT NULL. The same data is inserted in all four columns and is padded or truncated according to the datatype of the column.

```
create table zeros (bnot binary(5) not null,
            bnull binary(5) null,
            vnot varbinary(5) not null,
            vnull varbinary(5) null)

insert zeros values (0x12345000, 0x12345000,
            0x12345000, 0x12345000)
insert zeros values (0x123, 0x123, 0x123, 0x123)

select * from zeros
```

```
bnot             bnull        vnot         vnull
------------     ---------    ----------   ---------
0x1234500000     0x123450     0x123450     0x123450
0x0123000000     0x0123       0x0123       0x0123
```

Because each byte of storage holds 2 binary digits, Adaptive Server expects binary entries to consist of the characters "0x" followed by an even number of digits. When the "0x" is followed by an odd number of digits, Adaptive Server assumes that you omitted the leading 0 and adds it for you.

Input values "0x00" and "0x0" are stored as "0x00" in variable-length binary columns (*binary* **null**, *image* and *varbinary* columns). In fixed-length binary (*binary* **not null**) columns, the value is padded with zeros to the full length of the field:

```
insert zeros values (0x0, 0x0,0x0, 0x0)

select * from zeros where bnot = 0x00
```

```
bnot           bnull       vnot        vnull
----------     ------      -----       ------------
0x0000000000   0x00        0x00        0x00
```

If the input value does not include the "0x", Adaptive Server assumes that the value is an ASCII value and converts it. For example:

```
create table sample (col_a binary(8))

insert sample values ('002710000000ae1b')

select * from sample
```

```
col_a
------------------
0x3030323731303030
```

Platform Dependence

The exact form in which you enter a particular value depends upon the platform you are using. Therefore, calculations involving binary data can produce different results on different machines.

You cannot use the aggregate functions **sum** or **avg** with the binary datatypes.

For platform-independent conversions between hexadecimal strings and integers, use the **inttohex** and **hextoint** functions rather than the platform-specific convert function. For details, see "Datatype Conversion Functions" in Chapter 2, "Transact-SQL Functions.")

Standards and Compliance

Standard	Compliance Level
SQL92	The *binary* and *varbinary* datatypes are Transact-SQL extensions.

bit Datatypes

Function

Use the *bit* datatype for columns that contain true/false and yes/no types of data. The *status* column in the *syscolumns* system table indicates the unique offset position for *bit* datatype columns.

Entering Data into *bit* Columns

bit columns hold either 0 or 1. Integer values other than 0 or 1 are accepted, but are always interpreted as 1.

Storage Size

Storage size is 1 byte. Multiple *bit* datatypes in a table are collected into bytes. For example, 7 *bit* columns fit into 1 byte; 9 *bit* columns take 2 bytes.

Restrictions

Columns with a datatype of *bit* cannot be NULL and cannot have indexes on them.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

sysname Datatype

Function

sysname is a user-defined datatype that is distributed on the Adaptive Server installation tape and used in the system tables. Its definition is:

```
varchar(30) "not null"
```

Using the *sysname* Datatype

You cannot declare a column, parameter, or variable to be of type *sysname*. It is possible, however, to create a user-defined datatype with a base type of *sysname*. You can then define columns, parameters, and variables with the user-defined datatype.

Standards and Compliance

Standard	Compliance Level
SQL92	All user-defined datatypes, including *sysname*, are Transact-SQL extensions.

text and image Datatypes

Function

text columns are variable-length columns that can hold up to 2,147,483,647 (2^{31} - 1) bytes of printable characters.

image columns are variable-length columns that can hold up to 2,147,483,647 (2^{31} - 1) bytes of hexadecimal-like data.

Defining a *text* or *image* Column

You define a *text* or *image* column as you would any other column, with a **create table** or **alter table** statement. *text* and *image* datatype definitions do not include lengths. They do permit null values. The column definition takes the form:

column_name **{text | image} [null]**

For example, the **create table** statement for the author's *blurbs* table in the *pubs2* database with a *text* column, *blurb*, that permits null values, is:

```
create table blurbs
(au_id id not null,
copy text null)
```

To create the *au_pix* table in the *pubs2* database with an *image* column:

```
create table au_pix
(au_id          char(11) not null,
pic             image null,
format_type     char(11) null,
bytesize        int null,
pixwidth_hor    char(14) null,
pixwidth_vert   char(14) null)
```

How Adaptive Server Stores *text* and *image* Data

Adaptive Server stores *text* and *image* data in a linked list of data pages that are separate from the rest of the table. Each *text* or *image* page stores a maximum of 1800 bytes of data. All *text* and *image* data for a table is stored in a single page chain, regardless of the number of *text* and *image* columns the table contains.

Putting Additional Pages on Another Device

You can place subsequent *text* and *image* data pages on a different logical device with **sp_placeobject**.

Zero Padding

image values of less than 255 bytes that have an odd number of hexadecimal digits are padded with a leading zero (an insert of "0xaaabb" becomes "0x0aaabb").

➤ ***Note***

It is an error to insert *image* values of more than 255 bytes that have an odd number of bytes.

Partitioning Has No Effect on How the Data Is Stored

You can use the **partition** option of the **alter table** command to partition a table that contains *text* and *image* columns. Partitioning the table creates additional page chains for the other columns in the table, but has **no** effect on the way the *text* and *image* columns are stored.

Initializing text and image Columns

text and *image* columns are not initialized until you update them or insert a non-null value. Initialization allocates at least one data page for each non-null *text* or *image* data value. It also creates a pointer in the table to the location of the *text* or *image* data.

For example, the following statements create the table *testtext* and initialize the *blurb* column by inserting a non-null value. The column now has a valid text pointer, and the first 2K data page has been allocated.

```
create table texttest
(title_id varchar(6), blurb text null, pub_id
char(4))

insert texttest values
("BU7832", "Straight Talk About Computers is an
annotated analysis of what computers can do for
you: a no-hype guide for the critical user.",
"1389")
```

The following statements create a table for *image* values and initialize the *image* column:

```
create table imagetest
(image_id varchar(6), imagecol image null,
graphic_id char(4))
```

```
insert imagetest values
("94732", 0x0000008300000000000100000000013c,
"1389")
```

➤ ***Note***

Remember to surround *text* values with quotation marks and precede *image* values with the characters "0x".

For information on inserting and updating *text* and *image* data with Client-Library programs, see the *Client-Library/C Reference Manual*.

Saving Space by Allowing Nulls

To save storage space for empty *text* or *image* columns, define them to permit null values and **insert** nulls until you use the column. Inserting a null value does not initialize a *text* or *image* column and, therefore, does not create a text pointer or allocate 2K bytes of storage. For example, the following statement inserts values into the *title_id* and *pub_id* columns of the *testtext* table created above, but does not initialize the *blurb* text column:

```
insert texttest
(title_id, pub_id) values ("BU7832", "1389")
```

After a *text* or *image* row is given a non-null value, it always contains at least one data page. Resetting the value to null does not deallocate its data page.

Getting Information from *sysindexes*

Each table with *text* or *image* columns has an additional row in *sysindexes* that provides information about these columns. The *name* column in *sysindexes* uses the form "t*tablename*". The *indid* is always 255. These columns provide information about text storage:

Table 1-16: Storage of text and image data

Column	Description
ioampg	Pointer to the allocation page for the text page chain
first	Pointer to the first page of text data
root	Pointer to the last page
segment	Number of the segment where the object resides

You can query the *sysindexes* table for information about these columns. For example, the following query reports the number of data pages used by the *blurbs* table in the *pubs2* database:

```
select name, data_pgs(object_id("blurbs"), ioampg)
from sysindexes
where name = "tblurbs"

name
------------------------------ -----------
tblurbs                                  7
```

Using *readtext* and *writetext*

Before you can use **writetext** to enter *text* data or **readtext** to read it, you must initialize the *text* column. For details, see **readtext** and **writetext**.

Using **update** to replace existing *text* and *image* data with NULL reclaims all allocated data pages except the first page, which remains available for future use of **writetext**. To deallocate all storage for the row, use **delete** to remove the entire row.

Determining How Much Space a Column Uses

sp_spaceused provides information about the space used for text data as *index_size*:

```
sp_spaceused blurbs
```

```
name            rowtotal  reserved  data    index_size unused
--------------- --------  --------- ------- ---------- ------
blurbs          6         32 KB     2 KB    14 KB      16 KB
```

Restrictions on text and image Columns

text and *image* columns cannot be used:

- As parameters to stored procedures or as values passed to these parameters
- As local variables
- In **order by**, **compute**, **group by**, and **union** clauses
- In an index
- In subqueries or joins
- In a **where** clause, except with the keyword **like**
- With the + concatenation operator
- In the **if update** clause of a trigger

Selecting *text* and *image* Data

The following global variables return information on *text* and *image* data:

Table 1-17: text and image global variables

Variable	Explanation
@@textptr	The text pointer of the last *text* or *image* column inserted or updated by a process. Do not confuse this global variable with the Open Client **textptr()** function.
@@textcolid	ID of the column referenced by *@@textptr*.
@@textdbid	ID of a database containing the object with the column referenced by *@@textptr*.
@@textobjid	ID of the object containing the column referenced by *@@textptr*.
@@textsize	Current value of the **set textsize** option, which specifies the maximum length, in bytes, of *text* or *image* data to be returned with a **select** statement. It defaults to 32K. The maximum size for *@@textsize* is 231 - 1 (that is, 2,147,483,647).
@@textts	Text timestamp of the column referenced by *@@textptr*.

Converting the *text* and *image* Datatypes

You can explicitly convert *text* values to *char* or *varchar* and *image* values to *binary* or *varbinary* with the **convert** function, but you are limited to the maximum length of the character and binary datatypes, 255 bytes. If you do not specify the length, the converted value has a default length of 30 bytes. Implicit conversion is not supported.

Pattern Matching in *text* Data

Use the **patindex** function to search for the starting position of the first occurrence of a specified pattern in a *text*, *varchar*, or *char* column. The % wildcard character must precede and follow the pattern (except when you are searching for the first or last character).

You can also use the **like** keyword to search for a particular pattern. The following example selects each *text* data value from the *copy* column of the *blurbs* table that contains the pattern “Net Etiquette”.

```
select copy from blurb
where copy like "%Net Etiquette%"
```

Duplicate Rows Are Prohibited

The pointer to the *text* or *image* data uniquely identifies each row. Therefore, a table that contains *text* or *image* data cannot contain duplicate rows unless all *text* and *image* data is NULL. If this is the case, the pointer has not been initialized.

Standards and Compliance

Standard	Compliance Level
SQL92	The *text* and *image* datatypes are Transact-SQL extensions.

User-Defined Datatypes

Function

User-defined datatypes are built from the system datatypes and from the *sysname* user-defined datatype. After you create a user-defined datatype, you can use it to define columns, parameters, and variables. Objects that are created from user-defined datatypes inherit the rules, defaults, null type, and IDENTITY property of the user-defined datatype, as well as inheriting the defaults and null type of the system datatypes on which the user-defined datatype is based.

Creating Frequently Used Datatypes in the *model* Database

A user-defined datatype must be created in each database in which it will be used. It is a good practice to create frequently used types in the *model* database. These types are automatically added to each new database (including *tempdb*, which is used for temporary tables) as it is created.

Creating a User-Defined Datatype

Adaptive Server allows you to create user-defined datatypes, based on any system datatype, with the **sp_addtype** system procedure. You cannot create a user-defined datatype based on another user-defined datatype, such as *timestamp* or the *tid* datatype in the *pubs2* database.

The *sysname* datatype is an exception to this rule. Though *sysname* is a user-defined datatype, you can use it to build user-defined datatypes.

User-defined datatypes are database objects. Their names are case-sensitive and must conform to the rules for identifiers.

You can bind rules to user-defined datatypes with **sp_bindrule** and bind defaults with **sp_bindefault**.

By default, objects built on a user-defined datatype inherit the user-defined datatype's null type or IDENTITY property. You can override the null type or IDENTITY property in a column definition.

Renaming a User-Defined Datatype

Use **sp_rename** to rename a user-defined datatype.

Dropping a User-Defined Datatype

Use **sp_droptype** to remove a user-defined datatype from a database.

➤ ***Note***

You cannot drop a datatype that is already in use in a table.

Getting Help on Datatypes

Use the **sp_help** system procedure to display information about the properties of a system datatype or a user-defined datatype. You can also use **sp_help** to display the datatype, length, precision, and scale for each column in a table.

Standards and Compliance

Standard	Compliance Level
SQL92	User-defined datatypes are a Transact-SQL extension.

2 Transact-SQL Functions

This chapter describes the Transact-SQL functions. Functions are used to return information from the database. They are allowed in the **select** list, in the **where** clause, and anywhere an expression is allowed. They are often used as part of a stored procedure or program.

Types of Functions

Table 2-1 lists the different types of Transact-SQL functions and describes the type of information each returns.

Table 2-1: Types of Transact-SQL functions

Type of Function	Description
Aggregate Functions	Generate summary values that appear as new columns or as additional rows in the query results.
Datatype Conversion Functions	Change expressions from one datatype to another and specify new display formats for date/time information.
Date Functions	Do computations on *datetime* and *smalldatetime* values and their components, date parts.
Mathematical Functions	Return values commonly needed for operations on mathematical data.
Security Functions	Return security-related information.
String Functions	Operate on binary data, character strings, and expressions.
System Functions	Return special information from the database.
Text and Image Functions	Supply values commonly needed for operations on *text* and *image* data.

Table 2-2 lists the functions in alphabetical order.

Table 2-2: List of Transact-SQL functions

Function	Type	Return Value
abs	Mathematical	The absolute value of an expression.
acos	Mathematical	The angle (in radians) whose cosine is specified.
ascii	String	The ASCII code for the first character in an expression.

Table 2-2: List of Transact-SQL functions (continued)

Function	Type	Return Value
asin	Mathematical	The angle (in radians) whose sine is specified.
atan	Mathematical	The angle (in radians) whose tangent is specified.
atn2	Mathematical	The angle (in radians) whose sine and cosine are specified.
avg	Aggregate	The numeric average of all (distinct) values.
ceiling	Mathematical	The smallest integer greater than or equal to the specified value.
char	String	The character equivalent of an integer.
charindex	String	Returns an integer representing the starting position of an expression.
char_length	String	The number of characters in an expression.
col_length	System	The defined length of a column.
col_name	System	The name of the column whose table and column IDs are specified.
convert	Datatype Conversion	The specified value, converted to another datatype or a different *datetime* display format.
cos	Mathematical	The cosine of the specified angle (in radians).
cot	Mathematical	The cotangent of the specified angle (in radians).
count	Aggregate	The number of (distinct) non-null values.
curunreservedpgs	System	The number of free pages in the specified disk piece.
data_pgs	System	The number of pages used by the specified table or index.
datalength	System	The actual length, in bytes, of the specified column or string.
dateadd	Date	The date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.
datediff	Date	The difference between two dates.
datename	Date	The name of the specified part of a *datetime* value.
datepart	Date	The integer value of the specified part of a *datetime* value.
db_id	System	The ID number of the specified database.
db_name	System	The name of the database whose ID number is specified.
degrees	Mathematical	The size, in degrees, of an angle with a specified number of radians.
difference	String	The difference between two **soundex** values.

Table 2-2: List of Transact-SQL functions (continued)

Function	Type	Return Value
exp	Mathematical	The value that results from raising the constant e to the specified power.
floor	Mathematical	The largest integer that is less than or equal to the specified value.
getdate	Date	The current system date and time.
hextoint	Datatype Conversion	The platform-independent integer equivalent of the specified hexadecimal string.
host_id	System	The host process ID of the client process.
host_name	System	The current host computer name of the client process.
index_col	System	The name of the indexed column in the specified table or view.
inttohex	Datatype Conversion	The platform-independent, hexadecimal equivalent of the specified integer.
isnull	System	Substitutes the value specified in *expression2* when *expression1* evaluates to NULL.
is_sec_service_on	Security	"1" if the security service is active; "0" if it is not.
lct_admin	System	Manages the last-chance threshold.
license_enabled	System	"1" if the feature's license is enabled; "0" if it is not.
log	Mathematical	The natural logarithm of the specified number.
log10	Mathematical	The base 10 logarithm of the specified number.
lower	String	The uppercase equivalent of the specified expression.
isnull	String	The specified expression, trimmed of leading blanks.
max	Aggregate	The highest value in a column.
min	Aggregate	The lowest value in a column.
mut_excl_roles	System	The mutual exclusivity between two roles.
object_id	System	The object ID of the specified object.
object_name	System	The name of the object whose object ID is specified.
patindex	String, Text and Image	The starting position of the first occurrence of a specified pattern.
pi	Mathematical	The constant value 3.1415926535897936.
power	Mathematical	The value that results from raising the specified number to a given power.

Table 2-2: List of Transact-SQL functions (continued)

Function	Type	Return Value
proc_role	System	1 if the user has the correct role to execute the procedure; 0 if the user does not have this role.
ptn_data_pgs	System	The number of data pages used by a partition.
radians	Mathematical	The size, in radians, of an angle with a specified number of degrees.
rand	Mathematical	A random value between 0 and 1, generated using the specified seed value.
replicate	String	A string consisting of the specified expression repeated a given number of times.
reserved_pgs	System	The number of pages allocated to the specified table or index.
reverse	String	The specified string, with characters listed in reverse order.
right	String	The part of the character expression, starting the specified number of characters from the right.
role_contain	System	1 if *role2* contains *role1*.
role_id	System	The system role ID of the role whose name you specify.
role_name	System	The name of a role whose system role ID you specify.
round	Mathematical	The value of the specified number, rounded to a given number of decimal places.
rowcnt	System	An estimate of the number of rows in the specified table.
rtrim	String	The specified expression, trimmed of trailing blanks.
show_role	System	The login's currently active roles.
show_sec_services	Security	A list of the user's currently active security services.
sign	Mathematical	The sign (+1 for positive, 0, or -1 for negative) of the specified value.
sin	Mathematical	The sine of the specified angle (in radians).
soundex	String	A 4-character code representing the way an expression sounds.
space	String	A string consisting of the specified number of single-byte spaces.
sqrt	Mathematical	The square root of the specified number.
str	String	The character equivalent of the specified number.

Table 2-2: List of Transact-SQL functions (continued)

Function	Type	Return Value
stuff	String	The string formed by deleting a specified number of characters from one string and replacing them with another string.
substring	String	The string formed by extracting a specified number of characters from another string.
sum	Aggregate	The total of the values.
suser_id	System	The server user's ID number from the *syslogins* system table.
suser_name	System	The name of the current server user, or the user whose server user ID is specified.
tan	Mathematical	The tangent of the specified angle (in radians).
textptr	Text and Image	The pointer to the first page of the specified *text* column.
textvalid	Text and Image	1 if the pointer to the specified *text* column is valid; 0 if it is not.
tsequal	System	Compares *timestamp* values to prevent update on a row that has been modified since it was selected for browsing.
upper	String	The uppercase equivalent of the specified string.
used_pgs	System	The number of pages used by the specified table and its clustered index.
user	System	The name of the current server user.
user_id	System	The ID number of the specified user or the current user.
user_name	System	The name within the database of the specified user or the current user.
valid_name	System	0 if the specified string is not a valid identifier; a number other than 0 if the string is valid.
valid_user	System	1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

The following sections describe the types of functions in detail. The remainder of the chapter contains descriptions of the individual functions in alphabetical order.

Aggregate Functions

The aggregate functions generate summary values that appear as new columns in the query results. The aggregate functions are:

- **avg**
- **count**
- **max**
- **min**
- **sum**

Aggregate functions can be used in the select list or the **having** clause of a **select** statement or subquery. They cannot be used in a **where** clause.

Each aggregate in a query requires its own worktable. Therefore, a query using aggregates cannot exceed the maximum number of worktables allowed in a query (12).

When an aggregate function is applied to a *char* datatype value, it implicitly converts the value to *varchar*, stripping all trailing blanks.

Aggregates Used with *group* by

Aggregates are often used with **group by**. With **group by**, the table is divided into groups. Aggregates produce a single value for each group. Without **group by**, an aggregate function in the select list produces a single value as a result, whether it is operating on all the rows in a table or on a subset of rows defined by a **where** clause.

Aggregate Functions and NULL Values

Aggregate functions calculate the summary values of the non-null values in a particular column. If the **ansinull** option is set **off** (the default), there is no warning when an aggregate function encounters a null. If **ansinull** is set **on**, a query returns the following SQLSTATE warning when an aggregate function encounters a null:

```
Warning- null value eliminated in set function
```

Vector and Scalar Aggregates

Aggregate functions can be applied to all the rows in a table, in which case they produce a single value, a scalar aggregate. They can

also be applied to all the rows that have the same value in a specified column or expression (using the **group by** and, optionally, the **having** clause), in which case, they produce a value for each group, a vector aggregate. The results of the aggregate functions are shown as new columns.

You can nest a vector aggregate inside a scalar aggregate. For example:

```
select type, avg(price), avg(avg(price))
from titles
group by type
```

```
type
------------  ------------  ------------
UNDECIDED             NULL        15.23
business             13.73        15.23
mod_cook             11.49        15.23
popular_comp         21.48        15.23
psychology           13.50        15.23
trad_cook            15.96        15.23

(6 rows affected)
```

The **group by** clause applies to the vector aggregate—in this case, **avg**(*price*). The scalar aggregate, **avg**(**avg***(price)*), is the average of the average prices by type in the *titles* table.

In standard SQL, when a *select_list* includes an aggregate, all the *select_list* columns must either have aggregate functions applied to them or be in the **group by** list. Transact-SQL has no such restrictions.

Example 1 shows a **select** statement with the standard restrictions. Example 2 shows the same statement with another item (*title_id*) added to the select list. **order by** is also added to illustrate the difference in displays. These “extra” columns can also be referenced in a **having** clause.

```
1. select type, avg(price), avg(advance)
   from titles
   group by type
   type
   ------------  ------------  ------------
   UNDECIDED             NULL          NULL
   business             13.73     6,281.25
   mod_cook             11.49     7,500.00
   popular_comp         21.48     7,500.00
   psychology           13.50     4,255.00
   trad_cook            15.96     6,333.33

   (6 rows affected)

2. select type, title_id, avg(price), avg(advance)
   from titles
   group by type
   order by type
   type         title_id
   -----------  --------    ---------- ---------
   UNDECIDED       MC3026      NULL         NULL
   business        BU1032      13.73    6,281.25
   business        BU1111      13.73    6,281.25
   business        BU2075      13.73    6,281.25
   business        BU7832      13.73    6,281.25
   mod_cook        MC2222      11.49    7,500.00
   mod_cook        MC3021      11.49    7,500.00
   popular_comp    PC1035      21.48    7,500.00
   popular_comp    PC8888      21.48    7,500.00
   popular_comp    PC9999      21.48    7,500.00
   psychology      PS1372      13.50    4,255.00
   psychology      PS2091      13.50    4,255.00
   psychology      PS2106      13.50    4,255.00
   psychology      PS3333      13.50    4,255.00
   psychology      PS7777      13.50    4,255.00
   trad_cook       TC3218      15.96    6,333.33
   trad_cook       TC4203      15.96    6,333.33
   trad_cook       TC7777      15.96    6,333.33
```

You can use either a column name or any other expression (except a column heading or alias) after **group by**.

Null values in the **group by** column are put into a single group.

The **compute** clause in a **select** statement uses row aggregates to produce summary values. The row aggregates make it possible to retrieve detail and summary rows with one command. Example 3 illustrates this feature:

```
3. select type, title_id, price, advance
   from titles
   where type = "psychology"
   order by type
   compute sum(price), sum(advance) by type

type        title_id    price       advance
----------- -------     ----------  ---------
psychology  PS1372          21.59   7,000.00
psychology  PS2091          10.95   2,275.00
psychology  PS2106          7.00    6,000.00
psychology  PS3333          19.99   2,000.00
psychology  PS7777          7.99    4,000.00
                            sum     sum
                            ------- ----------
                            67.52   21,275.00
```

Note the difference in display between example 3 and the examples without **compute** (examples 1 and 2).

Aggregate functions cannot be used on virtual tables such as *sysprocesses* and *syslocks*.

If you include an aggregate function in the **select** clause of a cursor, that cursor cannot be updated.

Aggregate Functions As Row Aggregates

Row aggregate functions generate summary values that appear as additional rows in the query results.

To use the aggregate functions as row aggregates, use the following syntax:

Start of select statement

```
compute row_aggregate(column_name)
      [, row_aggregate(column_name)]...
   [by column_name [, column_name]...]
```

where:

- *column_name* is the name of a column. It must be enclosed in parentheses. Only exact numeric, approximate numeric, and money columns can be used with **sum** and **avg**.

  One **compute** clause can apply the same function to several columns. When using more than one function, use more than one **compute** clause.

- **by** indicates that row aggregate values are to be calculated for subgroups. Whenever the value of the **by** item changes, row aggregate values are generated. If you use **by**, you must use **order by**.

  Listing more than one item after **by** breaks a group into subgroups and applies a function at each level of grouping.

The row aggregates make it possible to retrieve detail and summary rows with one command. The aggregate functions, on the other hand, ordinarily produce a single value for all the selected rows in the table or for each group, and these summary values are shown as new columns.

The following examples illustrate the differences:

```
select type, sum(price), sum(advance)
from titles
where type like "%cook"
group by type

type
----------  ----------  ----------------
mod_cook         22.98         15,000.00
trad_cook        47.89         19,000.00

(2 rows affected)

select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type

type        price       advance
----------  ----------  ----------------
mod_cook          2.99         15,000.00
mod_cook         19.99              0.00
            sum         sum
            ----------  ----------------
                 22.98         15,000.00
type         price         advance
----------  ----------  ----------------
trad_cook        11.95          4,000.00
trad_cook        14.99          8,000.00
trad_cook        20.95          7,000.00
            sum         sum
            ----------  ----------------
                 47.89         19,000.00
(7 rows affected)
```

```
type        price       advance
----------  ----------  ----------------
mod_cook          2.99         15,000.00
mod_cook         19.99              0.00

Compute Result:
---------------------- -----------------
                 22.98         15,000.00
type         price         advance
----------  ----------  ----------------
trad_cook        11.95          4,000.00
trad_cook        14.99          8,000.00
trad_cook        20.95          7,000.00

Compute Result:
---------------------- -----------------
                 47.89         19,000.00
(7 rows affected)
```

The columns in the **compute** clause must appear in the select list.

If the **ansinull** option is set **off** (the default), there is no warning when a row aggregate encounters a null. If **ansinull** is set **on**, a query returns the following SQLSTATE warning when a row aggregate encounters a null:

```
Warning- null value eliminated in set function
```

You cannot use **select into** in the same statement as a **compute** clause because statements that include **compute** generate tables that include the summary results, which are not stored in the database.

Datatype Conversion Functions

Datatype conversion functions change expressions from one datatype to another and specify new display formats for date/time information. The datatype conversion functions are:

- **convert()**
- **inttohex()**
- **hextoint()**

The datatype conversion functions can be used in the select list, in the **where** clause, and anywhere else an expression is allowed.

Adaptive Server performs certain datatype conversions automatically. These are called **implicit conversions**. For example, if you compare a *char* expression and a *datetime* expression, or a *smallint*

expression and an *int* expression, or *char* expressions of different lengths, Adaptive Server automatically converts one datatype to another.

You must request other datatype conversions explicitly, using one of the built-in datatype conversion functions. For example, before concatenating numeric expressions, you must convert them to character expressions.

Adaptive Server does not allow you to convert certain datatypes to certain other datatypes, either implicitly or explicitly. For example, you cannot convert *smallint* data to *datetime* or *datetime* data to *smallint*. Unsupported conversions result in error messages.

Table 2-3 indicates whether individual datatype conversions are performed implicitly or explicitly or are unsupported.

Table 2-3: Explicit, implicit, and unsupported datatype conversions

From \ To	tinyint	smallint	int	decimal	numeric	real	float	char, nchar	varchar, nvarchar	text	smallmoney	money	bit	smalldatetime	datetime	binary	varbinary	image
tinyint	–	I	I	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
smallint	I	–	I	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
int	I	I	–	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
decimal	I	I	I	I/E	I/E	I	I	E	E	U	I	I	I	U	U	I	I	U
numeric	I	I	I	I/E	I/E	I	I	E	E	U	I	I	I	U	U	I	I	U
real	I	I	I	I	I	–	I	E	E	U	I	I	I	U	U	I	I	U
float	I	I	I	I	I	I	–	E	E	U	I	I	I	U	U	I	I	U
char, nchar	E	E	E	E	E	E	E	I	I	E	E	E	E	I	I	I	I	E
varchar, nvarchar	E	E	E	E	E	E	E	I	I	E	E	E	E	I	I	I	I	E
text	U	U	U	U	U	U	U	E	E	U	U	U	U	U	U	U	U	U
smallmoney	I	I	I	I	I	I	I	I	I	U	–	I	I	U	U	I	I	U
money	I	I	I	I	I	I	I	I	I	U	I	–	I	U	U	I	I	U
bit	I	I	I	I	I	I	I	I	I	U	I	I	–	U	U	I	I	U
smalldatetime	U	U	U	U	U	U	U	E	E	U	U	U	U	–	I	I	I	U
datetime	U	U	U	U	U	U	U	E	E	U	U	U	U	I	–	I	I	U
binary	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	–	I	E
varbinary	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I	–	E
image	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	E	E	U

Key:
E Explicit datatype conversion is required.
I Conversion can be done either implicitly or with an explicit datatype conversion function.
I/E Explicit datatype conversion function required when there is loss of precision or scale and arithabort numeric_truncation is on; otherwise, implicit conversion is allowed.
U Unsupported conversion.
– Conversion of a datatype to itself. These conversions are allowed but are meaningless.

Converting Character Data to a Non-Character Type

Character data can be converted to a non-character type—such as a money, date/time, exact numeric, or approximate numeric type—if it consists entirely of characters that are valid for the new type. Leading blanks are ignored. However, if a *char* expression that consists of a blank or blanks is converted to a *datetime* expression, SQL Server converts the blanks into the default *datetime* value of "Jan 1, 1900".

Syntax errors are generated when the data includes unacceptable characters. Following are some examples of characters that cause syntax errors:

- Commas or decimal points in integer data
- Commas in monetary data
- Letters in exact or approximate numeric data or bit stream data
- Misspelled month names in date/time data

Converting from One Character Type to Another

When converting from a multibyte character set to a single-byte character set, characters with no single-byte equivalent are converted to blanks.

text columns can be explicitly converted to *char, nchar, varchar,* or *nvarchar.* You are limited to the maximum length of the *character* datatypes, 255 bytes. If you do not specify the length, the converted value has a default length of 30 bytes.

Converting Numbers to a Character Type

Exact and approximate numeric data can be converted to a character type. If the new type is too short to accommodate the entire string, an insufficient space error is generated. For example, the following conversion tries to store a 5-character string in a 1-character type:

```
select convert(char(1), 12.34)
Insufficient result space for explicit conversion
of NUMERIC value '12.34' to a CHAR field.
```

➤ ***Note***

When converting *float* data to a character type, the new type should be at least 25 characters long.

Rounding During Conversion to and from Money Types

The *money* and *smallmoney* types store 4 digits to the right of the decimal point, but round up to the nearest hundredth (.01) for display purposes. When data is converted to a money type, it is rounded up to four places.

Data converted from a money type follows the same rounding behavior if possible. If the new type is an exact numeric with less than three decimal places, the data is rounded to the scale of the new type. For example, when $4.50 is converted to an integer, it yields 5:

```
select convert(int, $4.50)

 -----------
           5
```

Data converted to *money* or *smallmoney* is assumed to be in full currency units such as dollars rather than in fractional units such as cents. For example, the integer value of 5 is converted to the money equivalent of 5 dollars, not 5 cents, in the us_english language.

Converting Date/time Information

Data that is recognizable as a date can be converted to *datetime* or *smalldatetime*. Incorrect month names lead to syntax errors. Dates that fall outside the acceptable range for the datatype lead to arithmetic overflow errors.

When *datetime* values are converted to *smalldatetime*, they are rounded to the nearest minute.

Converting Between Numeric Types

Data can be converted from one numeric type to another. If the new type is an exact numeric whose precision or scale is not sufficient to hold the data, errors can occur.

For example, if you provide a float or numeric value as an argument to a built-in function that expects an integer, the value of the float or

numeric is truncated. However, Adaptive Server does not implicitly convert numerics that have a fractional part but returns a scale error message. For example, Adaptive Server returns error 241 for numerics that have a fractional part and error 257 if other datatypes are passed.

Use the **arithabort** and **arithignore** options to determine how Adaptive Server handles errors resulting from numeric conversions.

➤ ***Note***

The **arithabort** and **arithignore** options have been redefined for release 10.0 or later. If you use these options in your applications, examine them to be sure they are still producing the desired behavior.

Arithmetic Overflow and Divide-by-Zero Errors

Divide-by-zero errors occur when Adaptive Server tries to divide a numeric value by zero. Arithmetic overflow errors occur when the new type has too few decimal places to accommodate the results. This happens during:

- Explicit or implicit conversions to exact types with a lower precision or scale
- Explicit or implicit conversions of data that falls outside the acceptable range for a money or date/time type
- Conversions of hexadecimal strings requiring more than 4 bytes of storage using **hextoint**

Both arithmetic overflow and divide-by-zero errors are considered serious, whether they occur during an implicit or explicit conversion. Use the **arithabort arith_overflow** option to determine how Adaptive Server handles these errors. The default setting, **arithabort arith_overflow on**, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, **arithabort arith_overflow on** does not roll back earlier commands in the batch, and Adaptive Server does not execute statements that follow the error-generating statement in the batch. If you set **arithabort arith_overflow off**, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.You can use the *@@error* global variable to check statement results.

Use the **arithignore arith_overflow** option to determine whether Adaptive Server displays a message after these errors. The default setting, **off**,

displays a warning message when a divide-by-zero error or a loss of precision occurs. Setting **arithignore arith_overflow on** suppresses warning messages after these errors. The optional **arith_overflow** keyword can be omitted without any effect.

Scale Errors

When an explicit conversion results in a loss of scale, the results are truncated without warning. For example, when you explicitly convert a *float*, *numeric*, or *decimal* type to an *integer*, Adaptive Server assumes you want the result to be an integer and truncates all numbers to the right of the decimal point.

During implicit conversions to *numeric* or *decimal* types, loss of scale generates a scale error. Use the **arithabort numeric_truncation** option to determine how serious such an error is considered. The default setting, **arithabort numeric_truncation on**, aborts the statement that causes the error, but continues to process other statements in the transaction or batch. If you set **arithabort numeric_truncation off**, Adaptive Server truncates the query results and continues processing.

➤ ***Note***

For entry level SQL92 compliance, set:

- **arithabort arith_overflow off**
- **arithabort numeric_truncation on**
- **arithignore off**

Domain Errors

The **convert()** function generates a domain error when the function's argument falls outside the range over which the function is defined. This happens rarely.

Conversions Between Binary and Integer Types

The *binary* and *varbinary* types store hexadecimal-like data consisting of a "0x" prefix followed by a string of digits and letters.

These strings are interpreted differently by different platforms. For example, the string "0x0000100" represents 65536 on machines that consider byte 0 most significant and 256 on machines that consider byte 0 least significant.

Binary types can be converted to integer types either explicitly, using the **convert** function, or implicitly. If the data is too short for the new type, it is stripped of its "0x" prefix and zero-padded. If it is too long, it is truncated.

Both **convert** and the implicit datatype conversions evaluate binary data differently on different platforms. Because of this, results may vary from one platform to another. Use the **hextoint** function for platform-independent conversion of hexadecimal strings to integers, and the **inttohex** function for platform-independent conversion of integers to hexadecimal values.

Converting Between Binary and Numeric or Decimal Types

In *binary* and *varbinary* data strings, the first two digits after "0x" represent the *binary* type: "00" represents a positive number and "01" represents a negative number. When you convert a *binary* or *varbinary* type to *numeric* or *decimal*, be sure to specify the "00" or "01" values after the "0x" digit; otherwise, the conversion will fail.

For example, here is how to convert the following *binary* data to *numeric*:

```
select convert(numeric
(38, 18),0x00000000000000006b14bd1e6eea0000000000000000000000000000000000)

----------
123.456000
```

This example converts the same *numeric* data back to *binary*:

```
select convert(binary,convert(numeric(38, 18), 123.456))

------------------------------------------------------------
0x00000000000000006b14bd1e6eea0000000000000000000000000000000000
```

Converting Image Columns to Binary Types

You can use the **convert** function to convert an *image* column to *binary* or *varbinary*. You are limited to the maximum length of the *binary* datatypes, 255 bytes. If you do not specify the length, the converted value has a default length of 30 characters.

Converting Other Types to *bit*

Exact and approximate numeric types can be converted to the *bit* type implicitly. Character types require an explicit **convert** function.

The expression being converted must consist only of digits, a decimal point, a currency symbol, and a plus or minus sign. The presence of other characters generates syntax errors.

The *bit* equivalent of 0 is 0. The *bit* equivalent of any other number is 1.

Date Functions

The date functions manipulate values of the datatype *datetime* or *smalldatetime*.

The date functions are:

- **dateadd**
- **datediff**
- **datename**
- **datepart**
- **getdate**

Date functions can be used in the select list or **where** clause of a query.

Use the *datetime* datatype only for dates after January 1, 1753. *datetime* values must be enclosed in single or double quotes. Use *char, nchar, varchar* or *nvarchar* for earlier dates. Adaptive Server recognizes a wide variety of date formats. See "Datatype Conversion Functions" and "Date and Time Datatypes" in Chapter 1, "System and User-Defined Datatypes" for more information.

Adaptive Server automatically converts between character and *datetime* values when necessary (for example, when you compare a character value to a *datetime* value).

Date Parts

The date parts, the abbreviations recognized by Adaptive Server, and the acceptable values are:

Date Part	Abbreviation	Values
year	**yy**	1753 – 9999 (2079 for *smalldatetime*)
quarter	**qq**	1 – 4
month	**mm**	1 – 12
week	**wk**	1 – 54
day	**dd**	1 – 31
dayofyear	**dy**	1 – 366

Date Part	Abbreviation	Values
weekday	**dw**	1 – 7 (Sun.-Sat.)
hour	**hh**	0 – 23
minute	**mi**	0 – 59
second	**ss**	0 – 59
millisecond	**ms**	0 – 999

If you enter a year with only 2 digits, <50 is the next century ("25" is "2025") and >=50 is this century ("50" is "1950").

Milliseconds can be preceded either with a colon or a period. If preceded by a colon, the number means thousandths of a second. If preceded by a period, a single digit means tenths of a second, two digits mean hundredths of a second, and three digits mean thousandths of a second. For example, "12:30:20:1" means twenty and one-thousandth of a second past 12:30; "12:30:20.1" means twenty and one-tenth of a second past 12:30. Adaptive Server may round or truncate millisecond values when adding *datetime* data.

Mathematical Functions

Mathematical functions return values commonly needed for operations on mathematical data. Mathematical function names are not keywords.

Each function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric types also accept integer types. Adaptive Server automatically converts the argument to the desired type.

The mathematical functions are:

- **abs**
- **acos**
- **asin**
- **atan**
- **atn2**
- **ceiling**
- **cos**
- **cot**
- **degrees**
- **exp**

- **floor**
- **log**
- **log10**
- **pi**
- **power**
- **radians**
- **rand**
- **round**
- **sign**
- **sin**
- **sqrt**
- **tan**

Error traps are provided to handle domain or range errors of these functions. Users can **set** the **arithabort** and **arithignore** options to determine how domain errors are handled:

- **arithabort arith_overflow** specifies behavior following a divide-by-zero error or a loss of precision. The default setting, **arithabort arith_overflow on**, rolls back the entire transaction or aborts the batch in which the error occurs. If you set **arithabort arith_overflow off**, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.
- **arithabort numeric_truncation** specifies behavior following a loss of scale by an exact numeric type during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, **arithabort numeric_truncation on**, aborts the statement that causes the error, but continues to process other statements in the transaction or batch. If you set **arithabort numeric_truncation off**, Adaptive Server truncates the query results and continues processing.
- By default, the **arithignore arith_overflow** option is turned **off**, causing Adaptive Server to display a warning message after any query that results in numeric overflow. Set the **arithignore** option **on** to ignore overflow errors.

➤ ***Note***

The **arithabort** and **arithignore** options have been redefined for release 10.0 or later. If you use these options in your applications, examine them to be sure they still produce the desired effects.

Security Functions

Security functions return security-related information.

The security functions are:

- **is_sec_service_on**
- **show_sec_services**

String Functions

String function operate on binary data, character strings, and expressions. The string functions are:

- **ascii**
- **char**
- **charindex**
- **char_length**
- **difference**
- **lower**
- **ltrim**
- **patindex**
- **replicate**
- **reverse**
- **right**
- **rtrim**
- **soundex**
- **space**
- **str**
- **stuff**

- **substring**
- **upper**

String functions can be nested, and they can be used in a select list, in a **where** clause, or anywhere an expression is allowed. When you use constants with a string function, enclose them in single or double quotes. String function names are not keywords.

Each string function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric expressions also accept integer expressions. Adaptive Server automatically converts the argument to the desired type.

Limits on String Functions

Results of string functions are limited to 255 characters.

If **set string_rtruncation** is **on**, a user receives an error if an **insert** or **update** truncates a character string. However, SQL Server does not report an error if a displayed string is truncated. For example:

```
select replicate("a", 250) + replicate("B", 250)

------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaBBBBB
```

System Functions

System functions return special information from the database. The system functions are:

- **col_length**
- **col_name**
- **curunreservedpgs**
- **data_pgs**
- **datalength**
- **db_id**

- **db_name**
- **host_id**
- **host_name**
- **index_col**
- **isnull**
- **lct_admin**
- **mut_excl_roles**
- **object_id**
- **object_name**
- **proc_role**
- **ptn_data_pgs**
- **reserved_pgs**
- **role_contain**
- **role_id**
- **role_name**
- **rowcnt**
- **show_role**
- **suser_id**
- **suser_name**
- **tsequal**
- **used_pgs**
- **user**
- **user_id**
- **user_name**
- **valid_name**
- **valid_user**

The system functions can be used in a **select** list, in a **where** clause, and anywhere an expression is allowed.

When the argument to a system function is optional, the current database, host computer, server user, or database user is assumed.

Text and Image Functions

Text and image functions operate on *text* and *image* data. The text and image functions are:

- **textptr**
- **textvalid**

Text and image built-in function names are not keywords. Use the **set textsize** option to limit the amount of *text* or *image* data that is retrieved by a **select** statement.

The **patindex** text function can be used on *text* and *image* columns and can also be considered a text and image function.

Use the **datalength** function to get the length of data in *text* and *image* columns.

text and *image* columns cannot be used:

- As parameters to stored procedures
- As values passed to stored procedures
- As local variables
- In **order by**, **compute**, and **group by** clauses
- In an index
- In a **where** clause, except with the keyword **like**
- In joins
- In triggers

abs

Function

Returns the absolute value of an expression.

Syntax

abs(*numeric_expression*)

Arguments

numeric_expression – is a column, variable, or expression whose datatype is an exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types.

Examples

```
1. select abs(-1)

   -----------
             1
```

Returns the absolute value of -1.

Comments

- **abs**, a mathematical function, returns the absolute value of a given expression. Results are of the same type and have the same precision and scale as the numeric expression.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **abs**.

See Also

Function	**ceiling**, **floor**, **round**, **sign**

acos

Function

Returns the angle (in radians) whose cosine is specified.

Syntax

acos(*cosine*)

Arguments

cosine – is the cosine of the angle, expressed as a column name, variable, or constant of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

Examples

```
1. select acos(0.52)

--------------------
            1.023945
```

Returns the angle whose cosine is 0.52.

Comments

- **acos**, a mathematical function, returns the angle (in radians) whose cosine is the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **acos**.

See Also

Functions	**cos**, **degrees**, **radians**

ascii

Function

Returns the ASCII code for the first character in an expression.

Syntax

ascii(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select au_lname, ascii(au_lname) from authors
   where ascii(au_lname) < 70
```

```
 au_lname
 ------------------------------ -----------
 Bennet                                  66
 Blotchet-Halls                          66
 Carson                                  67
 DeFrance                                68
 Dull                                    68
```

Returns the authors last names and the ACSII codes for the first letters in their last names, if the ASCII code is less than 70.

Comments

- **ascii**, a string function, returns the ASCII code for the first character in the expression.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **ascii**.

See Also

Functions	**char**

asin

Function

Returns the angle (in radians) whose sine is specified.

Syntax

asin(*sine*)

Arguments

sine – is the sine of the angle, expressed as a column name, variable, or constant of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

Examples

```
1. select asin(0.52)

   --------------------
                0.546851
```

Comments

- **asin**, a mathematical function, returns the angle (in radians) whose sine is the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **asin**.

See Also

Functions	**degrees**, **radians**, **sin**

atan

Function

Returns the angle (in radians) whose tangent is specified.

Syntax

atan(*tangent*)

Arguments

tangent – is the tangent of the angle, expressed as a column name, variable, or constant of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

Examples

```
1. select atan(0.50)

   --------------------
                0.463648
```

Comments

- **atan**, a mathematical function, returns the angle (in radians) whose tangent is the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **atan**.

See Also

Functions	**atn2**, **degrees**, **radians**, **tan**

atn2

Function

Returns the angle (in radians) whose sine and cosine are specified.

Syntax

atn2(*sine*, *cosine*)

Arguments

sine – is the sine of the angle, expressed as a column name, variable, or constant of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

cosine – is the cosine of the angle, expressed as a column name, variable, or constant of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

Examples

```
1. select atn2(.50, .48)

   --------------------
                0.805803
```

Comments

- **atn2**, a mathematical function, returns the angle (in radians) whose sine and cosine are specified.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **atn2**.

See Also

Functions	**atan**, **degrees**, **radians**, **tan**

avg

Function

Returns the numeric average of all (distinct) values.

Syntax

avg([all | distinct] *expression*)

Arguments

all – applies **avg** to all values. **all** is the default.

distinct – eliminates duplicate values before **avg** is applied. **distinct** is optional.

expression – is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

1.
```
select avg(advance), sum(total_sales)
from titles
where type = "business"
```

```
----------------------- -----------
                6,281.25       30788
```

Calculates the average advance and the sum of total sales for all business books. Each of these aggregate functions produces a single summary value for all of the retrieved rows.

2.
```
select type, avg(advance), sum(total_sales)
from titles
group by type
```

```
type
 ------------ ---------------------- -----------
 UNDECIDED                      NULL        NULL
 business                   6,281.25       30788
 mod_cook                   7,500.00       24278
 popular_comp               7,500.00       12875
 psychology                 4,255.00        9939
 trad_cook                  6,333.33       19566
```

Used with a **group by** clause, the aggregate functions produce single values for each group, rather than for the whole table. This statement produces summary values for each type of book.

```
3. select pub_id, sum(advance), avg(price)
   from titles
   group by pub_id
   having sum(advance) > $25000 and avg(price) > $15
```

Groups the *titles* table by publishers and includes only those groups of publishers who have paid more than $25,000 in total advances and whose books average more than $15 in price.

```
pub_id
------ -------------------- --------------------
0877              41,000.00                15.41
1389              30,000.00                18.98
```

Comments

- **avg**, an aggregate function, finds the average of the values in a column. **avg** can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating averages.
- For general information about aggregate functions, see "Aggregate Functions" on page 2-6.
- When you average integer data, Adaptive Server treats the result as an *int* value, even if the datatype of the column is *smallint* or *tinyint*. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type *int*.
- You cannot use **avg**() with the binary datatypes.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **avg**.

See Also

Functions	**max**, **min**

ceiling

Function

Returns the smallest integer greater than or equal to the specified value.

Syntax

ceiling(*value*)

Arguments

value – is a column, variable, or expression whose datatype is exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types.

Examples

```
1. select ceiling(123.45)

   124

2. select ceiling(-123.45)

   -123

3. select ceiling(1.2345E2)

   24.000000

4. select ceiling(-1.2345E2)

   -123.000000

5. select ceiling($123.45)

   124.00

6. select discount, ceiling(discount) from
   salesdetail where title_id = "PS3333"

   discount
    -------------------- --------------------
               45.000000            45.000000
               46.700000            47.000000
               46.700000            47.000000
               50.000000            50.000000
```

Comments

- **ceiling**, a mathematical function, returns the smallest integer that is greater than or equal to the specified value. The return value has the same datatype as the value supplied.

For *numeric* and *decimal* values, results have the same precision as the value supplied and a scale of zero.

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **ceiling**.

See Also

Commands	**set**
Functions	**abs**, **floor**, **round**, **sign**

char

Function

Returns the character equivalent of an integer.

Syntax

char(*integer_expr*)

Arguments

integer_expr – is any integer (*tinyint*, *smallint*, or *int*) column name, variable, or constant expression between 0 and 255.

Examples

```
1. select char(42)

   -
   *

2. select xxx = char(65)

   xxx
   ---
   A
```

Comments

- **char**, a string function, converts a single-byte integer value to a character value. (**char** is usually used as the inverse of **ascii**.)
- **char** returns a *char* datatype. If the resulting value is the first byte of a multibyte character, the character may be undefined.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Reformatting Output with char

- You can use concatenation and **char**() values to add tabs or carriage returns to reformat output. **char**(10) converts to a return; **char**(9) converts to a tab.

For example:

```
/* just a space */
select title_id +  " " + title from titles where title_id = "T67061"
/* a return */
select title_id + char(10) + title from titles where title_id = "T67061"
/* a tab */
select title_id + char(9) + title from titles where title_id = "T67061"

------------------------------------------------------------------------
T67061 Programming with Curses
------------------------------------------------------------------------
T67061

Programming with Curses
------------------------------------------------------------------------
T67061      Programming with Curses
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **char.**

See Also

Functions	ascii, str

charindex

Function

Returns an integer representing the starting position of an expression.

Syntax

charindex(*expression1, expression2*)

Arguments

expression – is a binary or character column name, variable or constant expression. Can be *char*, *varchar*, *nchar* or *nvarchar* data, *binary* or *varbinary*.

Examples

```
1. select charindex("wonderful", notes)
   from titles
   where title_id = "TC3218"

   -----------
           46
```

Returns the position at which the character expression "wonderful" begins in the *notes* column of the *titles* table.

Comments

- **charindex**, a string function, searches *expression2* for the first occurrence of *expression1* and returns an integer representing its starting position. If *expression1* is not found, **charindex** returns 0.
- If *expression1* contains wildcard characters, **charindex** treats them as literals.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **charindex.**

See Also

Functions	patindex

char_length

Function

Returns the number of characters in an expression.

Syntax

char_length(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char, varchar, nchar* or *nvarchar* type.

Examples

```
1. select char_length(notes) from titles
   where title_id = "PC9999"

    -----------
             39

2. declare @var1 varchar(20), @var2 varchar(20),
   @char char(20)
   select @var1 = "abcd", @var2 = "abcd    ",
       @char = "abcd"
   select char_length(@var1), char_length(@var2),
   char_length(@char)

    ----------- ----------- -----------
              4           8          20
```

Comments

- **char_length**, a string function, returns an integer representing the number of characters in a character expression or text value.
- For variable-length columns and variables, **char_length** returns the number of characters (not the defined length of the column or variable). If explicit trailing blanks are included in variable-length variables, they are not stripped. For literals and fixed-length character columns and variables, **char_length** does not strip the expression of trailing blanks (see example 2).
- For multi-byte character sets, the number of characters in the expression is usually less than the number of bytes; use **datalength** to determine the number of bytes.
- If *char_expr* is NULL, **char_length** returns NULL.

- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **char_length.**

See Also

Functions	datalength

col_length

Function

Returns the defined length of a column.

Syntax

col_length(*object_name, column_name*)

Arguments

object_name – is name of a database object, such as a table, view, procedure, trigger, default, or rule. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.

column_name – is the name of the column.

Examples

```
1. select x = col_length("titles", "title")

    x
    ----
      80
```

Finds the length of the *title* column in the *titles* table. The "x" gives a column heading to the result.

Comments

- **col_length**, a system function, returns the defined length of column.
- For general information about system functions, see "System Functions" on page 2-23.
- To find the actual length of the data stored in each row, use **datalength**.
- For *text* and *image* columns, **col_length** returns 16, the length of the *binary(16)* pointer to the actual text page.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **col_length.**

See Also

Functions	datalength

col_name

Function

Returns the name of the column whose table and column IDs are specified.

Syntax

```
col_name(object_id, column_id[, database_id])
```

Arguments

object_id – is a numeric expression that is an object ID for a table, view, or other database object. These are stored in the *id* column of *sysobjects*.

column_id – is a numeric expression that is a column ID of a column. These are stored in the *colid* column of *syscolumns*.

database_id – is a numeric expression that is the ID for a database. These are stored in the *db_id* column of *sysdatabases*.

Examples

```
1. select col_name(208003772, 2)

   ------------------------------
   title
```

Comments

- **col_name**, a system function, returns the column's name.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **col_name.**

See Also

Functions	**db_id**, **object_id**

compare

Function

Allows you to directly compare two character strings based on alternate collation rules

Syntax

```
compare (char_expression1, char_expression2
   [, {collation_name | collation_ID}])
```

Arguments

char_expression1 is the character expression you want to compare to *char_expression2*.

char_expression2 is the character expression against which you want to compare *char_expression1*.

char_expression1 and *char_expression2* can be one of the following:

- Character type (*char*, *varchar*, *nchar*, or *nvarchar*)
- Character variable, or
- Constant character expression, enclosed in single or double quotation marks

collation_name can be a quoted string or a character variable that specifies the collation to use.

collation_ID is an integer constant or a variable that specifies the collation to use.

Comments

- The **compare** function returns the following values, based on the collation rules that you chose:
  - 1 – indicates that *char_expression1* is greater than *char_expression2*
  - 0 – indicates that *char_expression1* is equal to *char_expression2*
  - -1 – indicates that *char_expression1* is less than *char_expression2*
- Both *char_expression1* and *char_expression2* must be characters that are encoded in the server's default character set.
- Either *char_expression1* or *char_expression2*, or both, can be empty strings:

- If *char_expression2* is empty, the function returns 1.
- If both strings are empty, then they are equal, and the function returns a 0 value.
- If *char_expression1* is empty, the function returns a -1.

The **compare** function does not equate empty strings and strings containing only spaces, as Adaptive Server does. **compare** uses the **sortkey** function to generate collation keys for comparison. Therefore, a truly empty string, a string with one space, or a string with two spaces will not compare equally.

- If either *char_expression1* or *char_expression2* is NULL, then the result will be NULL.
- If you do not specify a value for *collation_name*, **compare** assumes binary collation.
- If you do not specify a value for *collation_ID*, **compare** assumes binary collation.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **compare.**

See Also

Functions	sortkey

convert

Function

Returns the specified value, converted to another datatype or a different *datetime* display format.

Syntax

```
convert (datatype [(length) | (precision[, scale])]
   [null | not null], expression [, style])
```

Arguments

datatype – is the system-supplied datatype (for example, *char*(10), *varbinary*(50), or *int*) into which to convert the expression. You cannot use user-defined datatypes.

When Java is enabled in the database, *datatype* can also be a Java-SQL class in the current database.

length – is an optional parameter used with *char, nchar, varchar, nvarchar, binary* and *varbinary* datatypes. If you do not supply a length, Adaptive Server truncates the data to 30 characters for the character types and 30 bytes for the binary types. The maximum allowable length for character and binary data is 255 bytes.

precision – is the number of significant digits in a *numeric* or *decimal* datatype. For *float* datatypes, precision is the number of significant binary digits in the mantissa. If you do not supply a precision, Adaptive Server uses the default precision of 18 for *numeric* and *decimal* datatypes.

scale – is the number of digits to the right of the decimal point in a *numeric*, or *decimal* datatype. If you do not supply a scale, Adaptive Server uses the default scale of 0.

null | **not null** – specifies the nullabilty of the result expression. If you do not supply either **null** or **not null**, the converted result has the same nullability as the expression.

expression – is the value to be converted from one datatype or date format to another.

When Java is enabled in the database, *expression* can be a value to be converted to a Java-SQL class.

style – is the display format to use for the converted data. When converting *money* or *smallmoney* data to a character type, use a *style* of 1 to display a comma after every 3 digits.

When converting *datetime* or *smalldatetime* data to a character type, use the style numbers in Table 2-4 to specify the display format. Values in the left-most column display 2-digit years (yy). For 4-digit years (yyyy), add 100, or use the value in the middle column.

Table 2-4: Display formats for date/time information

Without Century (yy)	With Century (yyyy)	Output
N / A	0 or 100	*mon dd yyyy hh:mi*AM (or PM)
1	101	*mm / dd / yy*
2	102	*yy.mm.dd*
3	103	*dd / mm / yy*
4	104	*dd.mm.yy*
5	105	*dd-mm-yy*
6	106	*dd mon yy*
7	107	*mon dd, yy*
8	108	*hh:mm:ss*
N / A	9 or 109	*mon dd yyyy hh:mi:ss:mmm*AM (or PM)
10	110	*mm-dd-yy*
11	111	*yy / mm / dd*
12	112	*yymmdd*

The default values (*style* 0 or 100), and *style* 9 or 109 always return the century (yyyy). When converting to *char* or *varchar* from *smalldatetime*, styles that include seconds or milliseconds show zeros in those positions.

Examples

```
1. select title, convert(char(12), total_sales)
   from titles

2. select title, total_sales
   from titles
   where convert(char(20), total_sales) like "1%"
```

3. **`select convert(char(12), getdate(), 3)`**

   Converts the current date to style "3", *dd/mm/yy*.

4. **`select convert(varchar(12), pubdate, 3) from titles`**

   If the value *pubdate* can be null, you must use *varchar* rather than *char*, or errors may result.

5. **`select convert(integer, 0x00000100)`**

   Returns the integer equivalent of the string "0x00000100". Results can vary from one platform to another.

6. **`select convert (binary, 10)`**

   Returns the platform-specific bit pattern as a Sybase binary type.

7. **`select convert(bit, $1.11)`**

   Returns 1, the bit string equivalent of $1.11.

8. **`select title, convert (char(100) not null, total_sales) into #tempsales from titles`**

   Creates *#tempsales* with *total_sales* of datatype *char(100)*, and does not allow null values. Even if *titles.total_sales* was defined as allowing nulls, *#tempsales* is created with *#tempsales.total_sales* not allowing null values.

Comments

- **convert**, a datatype conversion function, converts between a wide variety of datatypes and reformats date / time and money data for display purposes.
- For more information about datatype conversion, see "Datatype Conversion Functions" on page 2-11.
- **convert()** generates a domain error when the argument falls outside the range over which the function is defined. This should happen rarely.
- Use **null** or **not null** to specify the nullability of a target column. Specifically, this can be used with **select into** to create a new table and change the datatype and nullability of existing columns in the source table (See example 8, above).
- You can use **convert** to convert an *image* column to *binary* or *varbinary*. You are limited to the maximum length of the *binary* datatypes, 255 bytes. If you do not specify the length, the converted value has a default length of 30 characters.

Conversions Involving Java Classes

- When Java is enabled in the database, you can use **convert** to change datatypes in these ways:
  - Convert Java object types to SQL datatypes.
  - Convert SQL datatypes to Java types.
  - Convert any Java-SQL class installed in Adaptive Server to any other Java-SQL class installed in Adaptive Server if the compile-time datatype of the expression (the source class) is a subclass or superclass of the target class.

  The result of the conversion is associated with the current database.
- See *Java in Adaptive Server Enterprise* for a list of allowed datatype mappings and more information about datatype conversions involving Java classes.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **convert**.

See Also

Datatypes	User-Defined Datatypes
Functions	**hextoint**, **inttohex**

COS

Function

Returns the cosine of the specified angle.

Syntax

cos(*angle*)

Arguments

angle – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select cos(44)

    0.999843
```

Comments

- **cos**, a mathematical function, returns the cosine of the specified angle (in radians).
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **cos**.

See Also

Functions	**acos**, **degrees**, **radians**, **sin**

cot

Function

Returns the cotangent of the specified angle.

Syntax

cot(*angle*)

Arguments

angle – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select cot(90)

   --------------------
              -0.501203
```

Comments

- **cot**, a mathematical function, returns the cotangent of the specified angle (in radians).
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **cot**.

See Also

Functions	**degrees**, **radians**, **tan**

count

Function

Returns the number of (distinct) non-null values or the number of selected rows.

Syntax

```
count([all | distinct] expression)
```

Arguments

all – applies **count** to all values. **all** is the default.

distinct – eliminates duplicate values before **count** is applied. **distinct** is optional.

expression – is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

1.
```
select count(distinct city)
from authors
```

   Finds the number of different cities in which authors live.

2.
```
select type
from titles
group by type
having count(*) > 1
```

   Lists the types in the *titles* table, but eliminates the types that include only one book or none.

Comments

- **count**, an aggregate function, finds the number of non-null values in a column. For general information about aggregate functions, see "Aggregate Functions" on page 2-6.
- When **distinct** is specified, **count** finds the number of unique non-null values. **count** can be used with all datatypes except *text* and *image*. Null values are ignored when counting.

- **count**(*column_name*) returns a value of 0 on empty tables, on columns that contain only null values, and on groups that contain only null values.
- **count(*)** finds the number of rows. **count(*)** does not take any arguments, and cannot be used with **distinct**. All rows are counted, regardless of the presence of null values.
- When tables are being joined, include **count(*)** in the **select list** to produce the count of the number of rows in the joined results. If the objective is to count the number of rows from one table that match criteria, use **count**(*column_name*).
- **count()** can be used as an existence check in a subquery. For example:

  ```
  select * from tab where 0 <
      (select count(*) from tab2 where ...)
  ```

  However, because **count()** counts all matching values, **exists** or **in** may return results faster. For example:

  ```
  select * from tab where exists
      (select * from tab2 where ...)
  ```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **count**.

See Also

Commands	**compute Clause**, **group by and having Clauses**, **select**, **where Clause**

curunreservedpgs

Function

Returns the number of free pages in the specified disk piece.

Syntax

curunreservedpgs*(dbid, lstart, unreservedpgs*)

Arguments

dbid – is the ID for a database. These are stored in the *db_id* column of *sysdatabases*.

lstart – is a page within the disk piece for which pages are to be returned.

unreservedpgs – is the default value to return if the *dbtable* is presently unavailable for the requested database.

Examples

```
1. select db_name(dbid), d.name,
       curunreservedpgs(dbid, lstart, unreservedpgs)
       from sysusages u, sysdevices d
       where d.low <= u.size + vstart
           and d.high >= u.size + vstart -1
           and d.status &2 = 2
```

```
master           master        184
master           master        832
tempdb           master        464
tempdb           master       1016
tempdb           master        768
model            master        632
sybsystemprocs    master      1024
pubs2             master      248
```

Returns the database name, device name, and the number of unreserved pages for each device fragment.

```
2. select curunreservedpgs (dbid, sysusages.lstart, 0)
```

Displays the number of free pages on the segment for *dbid* starting on *sysusages.lstart*.

Comments

- **curunreservedpgs**, a system function, returns the number of free pages in a disk piece. For general information about system functions, see "System Functions" on page 2-23.
- If the database is open, the value is taken from memory; if the database is not in use, the value is taken from the *unreservedpgs* column in *sysusages*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **curunreservedpgs**.

See Also

Functions	**db_id**, **lct_admin**

data_pgs

Function

Returns the number of pages used by the specified table or index.

Syntax

```
data_pgs(object_id,
   {data_oam_pg_id | index_oam_pg_id})
```

Arguments

object_id – is an object ID for a table, view, or other database object. These are stored in the *id* column of *sysobjects*.

data_oam_pg_id – is the page ID for a data OAM page, stored in the *doampg* column of *sysindexes*.

index_oam_pg_id – is the page ID for an index OAM page, stored in the *ioampg* column of *sysindexes*.

Examples

1.
```
select sysobjects.name,
Pages = data_pgs(sysindexes.id, doampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
    and sysindexes.id > 100
    and (indid = 1 or indid = 0)
```

   Estimates the number of data pages used by user tables (which have object IDs that are greater than 100). An *indid* of 0 indicates a table without a clustered index; an *indid* of 1 indicates a table with a clustered index. This example does not include nonclustered indexes or text chains.

2.
```
select sysobjects.name,
Pages = data_pgs(sysindexes.id, ioampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
    and sysindexes.id > 100
    and (indid > 1)
```

   Estimates the number of data pages used by user tables (which have object IDs that are greater than 100), nonclustered indexes, and page chains.

Comments

- **data_pgs**, a system function, returns the number of pages used by a table (*doampg)* or index (*ioampg*). You must use this function in a query run against the *sysindexes* table. For more information on system functions, see “System Functions” on page 2-23.
- **data_pgs** works only on objects in the current database.
- The result does not include pages used for internal structures. To see a report of the number of pages for the table, clustered index, and internal structures, use **used_pgs**.

Accuracy of Results

- If used on the transaction log (*syslogs*), the result may not be accurate and can be off by up to 16 pages.

Errors

- Instead of returning an error, **data_pgs** returns 0 if any of the following are true:
  - The *object_id* does not exist in *sysobjects*
  - The *control_page_id* does not belong to the table specified by *object_id*
  - The *object_id* is -1
  - The *page_id* is -1

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **data_pgs**.

Tables Used

sysindexes, *syspartitions*

See Also

Functions	**object_id**, **rowcnt**, **used_pgs**
System procedures	**sp_spaceused**

datalength

Function

Returns the actual length, in bytes, of the specified column or string.

Syntax

datalength(*expression*)

Arguments

expression – is a column name, variable, constant expression, or a combination of any of these that evaluates to a single value. It can be of any datatype. *expression* is usually a column name. If *expression* is a character constant, it must be enclosed in quotes.

Examples

```
1. select Length = datalength(pub_name)
   from publishers
```

```
Length
-----------
         13
         16
          20
```

Finds the length of the *pub_name* column in the *publishers* table.

Comments

- **datalength**, a system function, returns the length of *expression* in bytes.
- **datalength** finds the actual length of the data stored in each row. **datalength** is useful on *varchar*, *varbinary*, *text* and *image* datatypes, since these datatypes can store variable lengths (and do not store trailing blanks). When a *char* value is declared to allow nulls, Adaptive Server stores it internally as a *varchar*. For all other datatypes, **datalength** reports their defined length.
- **datalength** of any NULL data returns NULL.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **datalength.**

See Also

Functions	**char_length**, **col_length**

dateadd

Function

Returns the date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.

Syntax

dateadd(*date_part*, *integer*, *date*)

Arguments

date_part – is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see "Date Parts" on page 2-19.

numeric – is an integer expression.

date – is either the function **getdate**, a character string in one of the acceptable date formats, an expression that evaluates to a valid date format, or the name of a *datetime* column.

Examples

```
1. select newpubdate = dateadd(day, 21, pubdate)
   from titles
```

Displays the new publication dates when the publication dates of all the books in the *titles* table slip by 21 days.

Comments

- **dateadd**, a date function, adds an interval to a specified date. For more information about date functions, see "Date Functions" on page 2-19.
- **dateadd** takes three arguments—the date part, a number, and a date. The result is a *datetime* value equal to the date plus the number of date parts.

  If the date argument is a *smalldatetime* value, the result is also a *smalldatetime*. You can use **dateadd** to add seconds or milliseconds to a *smalldatetime*, but it is meaningful only if the result date returned by **dateadd** changes by at least one minute.
- Use the *datetime* datatype only for dates after January 1, 1753. *datetime* values must be enclosed in single or double quotes. Use *char*, *nchar*, *varchar* or *nvarchar* for earlier dates. Adaptive Server

recognizes a wide variety of date formats. For more information, see "User-Defined Datatypes" in Chapter 1, "System and User-Defined Datatypes" and "Datatype Conversion Functions" in Chapter 2, "Transact-SQL Functions."

Adaptive Server automatically converts between character and *datetime* values when necessary (for example, when you compare a character value to a *datetime* value).

- Using the date part **weekday** or **dw** with **dateadd** is not logical, and produces spurious results. Use **day** or **dd** instead.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **dateadd.**

See Also

Commands	**select**, **where Clause**
Datatypes	"Date and Time Datatypes"
Functions	**datediff**, **datename**, **datepart**, **getdate**

datediff

Function

Returns the difference between two dates.

Syntax

datediff(*datepart*, *date1*, *date2*)

Arguments

datepart – is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see "Date Parts" on page 2-19.

date1 – can be either the function **getdate**, a character string in an acceptable date format, an expression that evaluates to a valid date format, or the name of a *datetime* column.

date2 – can be either the function **getdate**, a character string in an acceptable date format, an expression that evaluates to a valid date format, or the name of a *datetime* or *smalldatetime* column.

Examples

```
1. select newdate = datediff(day, pubdate, getdate())
   from titles
```

This query finds the number of days that have elapsed between *pubdate* and the current date (obtained with the **getdate** function).

Comments

- **datediff**, a date function, calculates the number of date parts between two specified dates. For more information about date functions, see "Date Functions" on page 2-19.
- **datediff** takes three arguments. The first is a date part. The second and third are dates. The result is a signed integer value equal to *date2* - *date1*, in date parts.
- **datediff** produces results of datatype *int*, and causes errors if the result is greater than 2,147,483,647. For milliseconds, this is approximately 24 days, 20:31.846 hours. For seconds, this is 68 years, 19 days, 3:14:07 hours.
- **datediff** results are always truncated, not rounded, when the result is not an even multiple of the date part. For example, using **hour** as

the date part, the difference between "4:00AM" and "5:50AM" is 1.

When you use **day** as the date part, **datediff** counts the number of midnights between the two times specified. For example, the difference between January 1, 1992, 23:00 and January 2, 1992, 01:00 is 1; the difference between January 1, 1992 00:00 and January 1, 1992, 23:59 is 0.

- The **month** datepart counts the number of first-of-the-months between two dates. For example, the difference between January 25 and February 2 is 1; the difference between January 1 and January 31 is 0.
- When you use the date part **week** with **datediff**, you get the number of Sundays between the two dates, including the second date but not the first. For example, the number of weeks between Sunday, January 4 and Sunday, January 11 is 1.
- If *smalldatetime* values are used, they are converted to *datetime* values internally for the calculation. Seconds and milliseconds in *smalldatetime* values are automatically set to 0 for the purpose of the difference calculation.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **datediff**.

See Also

Datatypes	"Date and Time Datatypes"
Commands	**select**, **where Clause**
Functions	**dateadd**, **datename**, **datepart**, **getdate**

datename

Function

Returns the name of the specified part of a *datetime* value.

Syntax

```
datename (datepart, date)
```

Arguments

datepart – is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see "Date Parts" on page 2-19.

date – can be either the function **getdate**, a character string in an acceptable date format, an expression that evaluates to a valid date format, or the name of a *datetime* or *smalldatetime* column.

Examples

```
1. select datename(month, getdate())
```

```
November
```

This example assumes a current date of November 20, 1998.

Comments

- **datename**, a date function, returns the name of the specified part (such as the month "June") of a *datetime* or *smalldatetime* value, as a character string. If the result is numeric, such as "23" for the day, it is still returned as a character string.
- For more information about date functions, see "Date Functions" on page 2-19.
- The date part **weekday** or **dw** returns the day of the week (Sunday, Monday, and so on) when used with **datename**.
- Since *smalldatetime* is accurate only to the minute, when a *smalldatetime* value is used with **datename**, seconds and milliseconds are always 0.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **datename.**

See Also

Datatypes	"Date and Time Datatypes"
Commands	**select**, **where Clause**
Functions	**dateadd**, **datediff**, **datepart**, **getdate**

datepart

Function

Returns the integer value of the specified part of a *datetime* value.

Syntax

datepart(*date_part*, *date*)

Arguments

date_part – is a date part. Table 2-5 lists the date parts, the abbreviations recognized by **datepart**, and the acceptable values.

Table 2-5: Date parts and their values

Date Part	Abbreviation	Values
year	**yy**	1753 – 9999 (2079 for *smalldatetime*)
quarter	**qq**	1 – 4
month	**mm**	1 – 12
week	**wk**	1 – 54
day	**dd**	1 – 31
dayofyear	**dy**	1 – 366
weekday	**dw**	1 – 7 (Sun.-Sat.)
hour	**hh**	0 – 23
minute	**mi**	0 – 59
second	**ss**	0 – 59
millisecond	**ms**	0 – 999
calweekofyear	**cwk**	1-53
calyearofweek	**cyr**	1753 – 9999
caldayofweek	**cdw**	1 – 7

If you enter the year as two digits, <50 is the next century (“25” is “2025”) and >=50 is this century (“50” is “1950”).

Milliseconds can be preceded by either a colon or a period. If preceded by a colon, the number means thousandths of a second. If preceded by a period, a single digit means tenths of a second, two digits mean hundredths of a second, and three digits mean thousandths of a second. For example, “12:30:20:1” means twenty and one-thousandth of a second past 12:30; “12:30:20.1” means twenty and one-tenth of a second past 12:30.

date – can be either the function **getdate**, a character string in an acceptable date format, an expression that evaluates to a valid date format, or the name of a *datetime* or *smalldatetime* column.

Examples

```
1. select datepart(month, getdate())

   -----------
            11
```

This example assumes a current date of November 25, 1995.

```
2. select datepart(year, pubdate) from titles where
   type = "trad_cook"

    -----------
           1990
           1985
           1987

3. select datepart(cwk,'1993/01/01')

   -----------
            53

4. select datepart(cyr,'1993/01/01')

   -----------
           1992

5. select datepart(cdw,'1993/01/01')

   -----------
             5
```

Comments

- **datepart**, a date function, returns an integer value for the specified part of a *datetime* value. For more information about date functions, see "Date Functions" on page 2-19.
- **datepart** returns a number that follows ISO standard 8601, which defines the first day of the week and the first week of the year. Depending on whether the **datepart** function includes a value for **calweekofyear, calyearofweek**, or **caldayorweek,** the date returned may be different for the same unit of time. For example, if Adaptive Server is configured to use US English as the default language:

  datepart(cyr, "1/1/1989")

  returns 1988, but:

  datepart(yy, "1/1/1989)

  returns 1989.

  This disparity occurs because the ISO standard defines the first week of the year as the first week that includes a Thursday **and** begins with Monday.

For servers using US English as their default language, the first day of the week as Sunday, and the first week of the year is the week that contains January 4th.

- The date part **weekday** or **dw** returns the corresponding number when used with **datepart**. The numbers that correspond to the names of weekdays depend on the **datefirst** setting. Some language defaults (including us_english) produce Sunday=1, Monday=2, and so on; others produce Monday=1, Tuesday=2, and so on.The default behavior can be changed on a per-session basis with **set datefirst**.
- **calweekofyear**, which can be abbreviated as **cwk**, returns the ordinal position of the week within the year. **calyearofweek**, which can be abbreviated as **cyr**, returns the year in which the week begins. **caldayofweek**, which can abbreviated as **cdw**, returns the ordinal position of the day within the week. You cannot use **calweekofyear**, **calyearofweek**, and **caldayofweek** as date parts for **dateadd**, **datediff** and **datename**.
- Since *smalldatetime* is accurate only to the minute, when a *smalldatetime* value is used with **datepart**, seconds and milliseconds are always 0.
- The values of the weekday date part are affected by the language setting.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **datepart**.

See Also

Datatypes	"Date and Time Datatypes"
Commands	**select**, **where Clause**
Functions	**dateadd**, **datediff**, **datename**, **getdate**

db_id

Function

Returns the ID number of the specified database.

Syntax

db_id(*database_name*)

Arguments

database_name – is the name of a database. *database_name* must be a character expression. If it is a constant expression, it must be enclosed in quotes.

Examples

```
1. select db_id("sybsystemprocs")

   ------
   4
```

Comments

- **db_id**, a system function, returns the database ID number.
- If you do not specify a *database_name*, **db_id** returns the ID number of the current database.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **db_id**.

See Also

Functions	**db_name**, **object_id**

db_name

Function

Returns the name of the database whose ID number is specified.

Syntax

db_name([*database_id*])

Arguments

database_id – is a numeric expression for the database ID (stored in *sysdatabases.dbid*).

Examples

1. select db_name()

Returns the name of the current database.

2. select db_name(4)

```
------------------------------
sybsystemprocs
```

Comments

- **db_name**, a system function, returns the database name.
- If no *database_id* is supplied, **db_name** returns the name of the current database.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **db_name**.

See Also

Functions	**db_id**, **col_name**, **object_name**

degrees

Function

Returns the size, in degrees, of an angle with the specified number of radians.

Syntax

degrees(*numeric*)

Arguments

numeric – is a number, in radians, to convert to degrees.

Examples

```
1. select degrees(45)

   -----------
          2578
```

Comments

- **degrees**, a mathematical function, converts radians to degrees. Results are of the same type as the numeric expression.

  For numeric and decimal expressions, the results have an internal precision of 77 and a scale equal to that of the expression.

  When money datatypes are used, internal conversion to *float* may cause loss of precision.

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **degrees**.

See Also

Functions	radians

difference

Function

Returns the difference between two **soundex** values.

Syntax

difference(*char_expr1, char_expr2*)

Arguments

char_expr1 – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

char_expr2 – is another character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select difference("smithers", "smothers")

   ---------
   4

2. select difference("smothers", "brothers")

   ---------
   2
```

Comments

- **difference**, a string function, returns an integer representing the difference between two **soundex** values.
- The **difference** function compares two strings and evaluates the similarity between them, returning a value from 0 to 4. The best match is 4.

  The string values must be composed of a contiguous sequence of valid single- or double-byte roman letters.
- If *char_expr1* or *char_expr2* is NULL, returns NULL.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **difference.**

See Also

Functions	soundex

exp

Function

Returns the value that results from raising the constant e to the specified power.

Syntax

exp(*approx_numeric*)

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select exp(3)

   --------------------
              20.085537
```

Comments

- **exp**, a mathematical function, returns the exponential value of the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **exp**.

See Also

Functions	**log**, **log10**, **power**

floor

Function

Returns the largest integer that is less than or equal to the specified value.

Syntax

floor(*numeric*)

Arguments

numeric – is any exact numeric (*numeric*, *dec*, *decimal*, *tinyint*, *smallint*, or *int*), approximate numeric (*float*, *real*, or *double precision*), or *money* column, variable, constant expression, or a combination of these.

Examples

```
1. select floor(123)
   -----------
           123
2. select floor(123.45)
   -------
       123
3. select floor(1.2345E2)
   --------------------
            123.000000
4. select floor(-123.45)
   -------
      -124
5. select floor(-1.2345E2)
   --------------------
           -124.000000
6. select floor($123.45)
   ------------------------
                    123.00
```

Comments

- **floor**, a mathematical function, returns the largest integer that is less than or equal to the specified value. Results are of the same type as the numeric expression.

  For numeric and decimal expressions, the results have a precision equal to that of the expression and a scale of 0.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **floor**.

See Also

Functions	**abs**, **ceiling**, **round**, **sign**

getdate

Function

Returns the current system date and time.

Syntax

```
getdate()
```

Arguments

None.

Examples

```
1. select getdate()
```

```
Nov 25 1995 10:32AM
```

```
2. select datepart(month, getdate())
```

```
1
```

```
3. select datename(month, getdate())
```

```
November
```

These examples assume a current date of November 25, 1995, 10:32 a.m.

Comments

- **getdate**, a date function, returns the current system date and time.
- For more information about date functions, see “Date Functions” on page 2-19.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **getdate**.

See Also

Datatypes	“Date and Time Datatypes”
Functions	**dateadd**, **datediff**, **datename**, **datepart**

hextoint

Function

Returns the platform-independent integer equivalent of a hexadecimal string.

Syntax

hextoint (*hexadecimal_string*)

Arguments

hexadecimal_string – is the hexadecimal value to be converted to an integer. This must be either a character type column or variable name or a valid hexadecimal string, with or without a "0x" prefix, enclosed in quotes.

Examples

```
1. select hextoint ("0x00000100")
```

Returns the integer equivalent of the hexadecimal string "0x00000100". The result is always 256, regardless of the platform on which it is executed.

Comments

- **hextoint**, a datatype conversion function, returns the platform-independent integer equivalent of a hexadecimal string.
- Use the **hextoint** function for platform-independent conversions of hexadecimal data to integers. **hextoint** accepts a valid hexadecimal string, with or without a "0x" prefix, enclosed in quotes, or the name of a character type column or variable.

  hextoint returns the integer equivalent of the hexadecimal string. The function always returns the same integer equivalent for a given hexadecimal string, regardless of the platform on which it is executed.
- For more information about datatype conversion, see "Datatype Conversion Functions" on page 2-11.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **hextoint.**

See Also

Functions	**convert**, **inttohex**

host_id

Function

Returns the host process ID or the client process.

Syntax

```
host_id()
```

Arguments

None.

Examples

```
1. select host_id()

   ------------------------------
   24711
```

Comments

- **host_id**, a system function, returns the host process ID of the client process (not the Server process).
- For general information about system functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **host_id**.

See Also

Functions	host_name

host_name

Function

Returns the current host computer name of the client process.

Syntax

```
host_name()
```

Arguments

None.

Examples

```
1. select host_name()

   ------------------------------
   violet
```

Comments

- **host_name**, a system function, returns the current host computer name of the client process (not the Server process).
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **host_name**.

See Also

Functions	host_id

index_col

Function

Returns the name of the indexed column in the specified table or view.

Syntax

```
index_col (object_name, index_id, key_# [, user_id])
```

Arguments

object_name – is the name of a table or view. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.

index_id – is the number of *object_name*'s index. This number is the same as the value of *sysindexes.indid*.

key_# – is a key in the index. This value is between 1 and *sysindexes.keycnt* for a clustered index and between 1 and *sysindexes.keycnt*+1 for a nonclustered index.

user_id – is the owner of *object_name*. If you do not specify *user_id*, it defaults to the caller's user ID.

Examples

1.
```
declare @keycnt integer
select @keycnt = keycnt from sysindexes
     where id = object_id("t4")
     and indid = 1
while @keycnt > 0
begin
     select index_col("t4", 1, @keycnt)
     select @keycnt = @keycnt - 1
end
```

Finds the names of the keys in the clustered index on table *t4*.

Comments

- **index_col**, a system function, returns the name of the indexed column.
- **index_col** returns NULL if *object_name* is not a table or view name.
- For general information about system functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **index_col.**

See Also

Functions	object_id
System Procedures	sp_helpindex

index_colorder

Function

Returns the column order.

Syntax

```
index_colorder (object_name, index_id, key_#
   [, user_id])
```

Arguments

object_name – is the name of a table or view. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.

index_id – is the number of *object_name*'s index. This number is the same as the value of *sysindexes.indid*.

key_# – is a key in the index. Valid values are 1 and the number of keys in the index. The number of keys is stored in *sysindexes.keycnt*.

user_id – is the owner of *object_name*. If you do not specify *user_id*, it defaults to the caller's user ID.

Examples

```
1. select name, index_colorder("sales", indid, 2)
   from sysindexes
   where id = object_id ("sales")
   and indid > 0

   name
   ------------------------ ------------------------
   salesind                 DESC
```

Returns "DESC" because the *salesind* index on the *sales* table is in descending order.

Comments

- **index_colorder**, a system function, returns "ASC" for columns in ascending order or "DESC" for columns in descending order.
- **index_colorder** returns NULL if *object_name* is not a table name or if *key_#* is not a valid key number.

- For general information about system functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **index_colorder.**

inttohex

Function

Returns the platform-independent hexadecimal equivalent of the specified integer.

Syntax

inttohex (*integer_expression*)

Arguments

integer_expression – is the integer value to be converted to a hexadecimal string.

Examples

```
1. select inttohex (10)

   --------
   0000000A
```

Comments

- **inttohex**, a datatype conversion function, returns the platform-independent hexadecimal equivalent of an integer, without a "0x" prefix.
- Use the **inttohex** function for platform-independent conversions of integers to hexadecimal strings. **inttohex** accepts any expression that evaluates to an integer. It always returns the same hexadecimal equivalent for a given expression, regardless of the platform on which it is executed.
- For more information about datatype conversion, see "Datatype Conversion Functions" on page 2-11.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **inttohex.**

See Also

Functions	**convert**, **hextoint**

isnull

Function

Substitutes the value specified in *expression2* when *expression1* evaluates to NULL.

Syntax

```
isnull(expression1, expression2)
```

Arguments

expression – is a column name, variable, constant expression, or a combination of any of these that evaluates to a single value. It can be of any datatype. *expression* is usually a column name. If *expression* is a character constant, it must be enclosed in quotes.

Examples

1.
```
select isnull(price,0)
from titles
```

   Returns all rows from the *titles* table, replacing null values in *price* with 0.

Comments

- **isnull**, a system function, substitutes the value specified in *expression2* when *expression1* evaluates to NULL. For general information about system functions, see "String Functions" on page 2-22.
- The datatypes of the expressions must convert implicitly, or you must use the **convert** function.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **isnull**.

See Also

Functions	convert

is_sec_service_on

Function

Returns 1 if the security service is active and 0 if it is not.

Syntax

is_sec_service_on(*security_service_nm*)

Arguments

security_service_nm – is the name of the security service.

Examples

```
1. select is_sec_service_on("unifiedlogin")
```

Comments

- Use **is_sec_service_on** to determine whether a given security service is active during the session.
- To find valid names of security services, run this query:

  ```
  select * from syssecmechs
  ```

  The *available_service* column displays the security services that are supported by Adaptive Server.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **is_sec_service_on**.

See Also

Functions	show_sec_services

lct_admin

Function

Manages the last-chance threshold.

Returns the current value of the last-chance threshold.

Aborts transactions in a transaction log that has reached its last-chance threshold.

Syntax

```
lct_admin({{"lastchance" | "logfull" }, database_id
   |"reserve", {log_pages | 0 }
   | "abort", process-id [, database-id]})
```

Arguments

lastchance – creates a last-chance threshold in the specified database.

logfull – returns 1 if the last-chance threshold has been crossed in the specified database and 0 if it has not.

database_id – specifies the database.

reserve – obtains either the current value of the last-chance threshold or the number of log pages required for dumping a transaction log of a specified size.

log_pages – is the number of pages for which to determine a last-chance threshold.

0 – returns the current value of the last-chance threshold. The size of the last-chance threshold in a database with separate log and data segments does not vary dynamically. It has a fixed value, based on the size of the transaction log. The last-chance threshold varies dynamically in a database with mixed log and data segments.

abort – aborts transactions in a database where the transaction log has reached its last-chance threshold. Only transactions in LOG SUSPEND mode can be aborted.

process-id – The ID (*spid*) of a process in log-suspend mode. A process is placed in log-suspend mode when it has open transactions in a transaction log that has reached its last-chance threshold (LCT).

database-id – the ID of a database whose transaction log has reached its LCT. If *process-id* is **0**, all open transactions in the specified database are terminated.

Examples

1. **`select lct_admin("lastchance", 1)`**

   This creates the log segment last-chance threshold for the database with *dbid* 1. It returns the number of pages at which the new threshold resides. If there was a previous last-chance threshold, it is replaced.

2. **`select lct_admin("logfull", 6)`**

   Returns 1 if the last-chance threshold for the database with *db_id* of 6 has been crossed, and 0 if it has not.

3. **`select lct_admin("reserve", 64)`**

   ```
   -----------
            16
   ```

   Calculates and returns the number of log pages that would be required to successfully dump the transaction log in a log containing 64 pages.

4. **`select lct_admin("reserve",0)`**

   Returns the current last-chance threshold of the transaction log in the database from which the command was issued.

5. **`select lct_admin("abort", 83)`**

   Aborts transactions belonging to process 83. The process must be in log-suspend mode. Only transactions in a transaction log that has reached its LCT are terminated.

6. **`select lct_admin("abort", 0, 5)`**

   Aborts all open transactions in the database with database ID 5.

   This form awakens any processes that may be suspended at the log segment last-chance threshold.

Comments

- **lct_admin**, a system function, manages the log segment's last-chance threshold. For general information about system functions, see "String Functions" on page 2-22.
- If **lct_admin("lastchance"**, *dbid***)** returns zero, the log is not on a separate segment in this database, so no last-chance threshold exists.

- Whenever you create a database with a separate log segment, the server creates a default last chance threshold that defaults to calling **sp_thresholdaction**. This happens even if a procedure called **sp_thresholdaction** does not exist on the server at all.

  If your log crosses the last-chance threshold, Adaptive Server suspends activity, tries to call **sp_thresholdaction**, finds it does not exist, generates an error, then leaves processes suspended until the log can be truncated.
- To terminate the oldest open transaction in a transaction log that has reached its LCT, enter the ID of the process that initiated the transaction.
- To terminate all open transactions in a transaction log that has reached its LCT, enter **0** as the *process_id*, and specify a database ID in the *database-id* parameter.
- For more information, see the *System Administration Guide*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Administrator can execute **lct_admin abort**. Any user can execute the other **lct_admin** options.

See Also

Commands	**dump transaction**
Functions	**curunreservedpgs**
System Procedures	**sp_addthreshold**, **sp_dropthreshold**, **sp_helpthreshold**, **sp_modifythreshold**, **sp_thresholdaction**

license_enabled

Function

Returns 1 if a feature's license is enabled, 0 if the license is not enabled, or null if you specify an invalid license name.

Syntax

```
license_enabled("ase_server" | "ase_ha" | "ase_dtm" |
   "ase_java" | "ase_asm")
```

Arguments

ase_server – specifies the license for Adaptive Server.

ase_ha – specifies the license for the Adaptive Server high availability feature.

ase_dtm – specifies the license for Adaptive Server distributed transaction management features.

ase_java – specifies the license for the Adaptive Server Java feature.

ase_asm – specifies the license for Adaptive Server advanced security mechanism.

Examples

```
1. select license_enabled("ase_dtm")

   --------------
                1
```

Indicates that the license for the Adaptive Server distributed transaction management feature is enabled.

Comments

- For information about installing license keys for Adaptive Server features, see your *Installation Guide.*

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **log**.

See Also

System Procedures	sp_configure

log

Function

Returns the natural logarithm of the specified number.

Syntax

log(*approx_numeric*)

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select log(20)

   --------------------
                2.995732
```

Comments

- **log**, a mathematical function, returns the natural logarithm of the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **log**.

See Also

Functions	**log10**, **power**

log10

Function

Returns the base 10 logarithm of the specified number.

Syntax

log10(*approx_numeric*)

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select log10(20)

   --------------------
             1.301030
```

Comments

- **log10**, a mathematical function, returns the base 10 logarithm of the specified value.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **log10**.

See Also

Functions	log, power

lower

Function

Returns the lowercase equivalent of the specified expression.

Syntax

lower(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select lower(city) from publishers

   --------------------
   boston
   washington
   berkeley
```

Comments

- **lower**, a string function, converts uppercase to lowercase, returning a character value.
- **lower** is the inverse of **upper**.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **lower**.

See Also

Functions	upper

ltrim

Function

Returns the specified expression, trimmed of leading blanks.

Syntax

```
ltrim(char_expr)
```

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select ltrim("    123")

   -------
   123
```

Comments

- **ltrim**, a string function, removes leading blanks from the character expression. Only values equivalent to the space character in the current character set are removed.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **ltrim**.

See Also

Functions	rtrim

max

Function

Returns the highest value in an expression.

Syntax

max(*expression*)

Arguments

expression – is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery.

Examples

```
1. select max(discount) from salesdetail

   --------------------
               62.200000
```

Returns the maximum value in the *discount* column of the *salesdetail* table as a new column.

```
2. select discount from salesdetail
   compute max(discount)
```

Returns the maximum value in the *discount* column of the *salesdetail* table as a new row.

Comments

- **max**, an aggregate function, finds the maximum value in a column or expression. For general information about aggregate functions, see "Aggregate Functions" on page 2-6.
- **max** can be used with exact and approximate numeric, character, and *datetime* columns. It cannot be used with *bit* columns. With character columns, **max** finds the highest value in the collating sequence. **max** ignores null values. **max** implicitly converts *char* datatypes to *varchar*, stripping all trailing blanks.
- Adaptive Server goes directly to the end of the index to find the last row for **max** when there is an index on the aggregated column, unless:
  - The *expression* not a column
  - The column is not the first column of an index

- There is another aggregate in the query
- There is a **group by** or **where** clause

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	By default, **max** is not compliant; use **set ansinull on** for compliant behavior.

Permissions

Any user can execute **max**.

See Also

Commands	**compute Clause**, **group by and having Clauses**, **select**, **where Clause**
Functions	**avg**, **min**

min

Function

Returns the lowest value in a column.

Syntax

min(*expression*)

Arguments

expression – is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

```
1. select min(price) from titles
   where type = "psychology"

   -----------------------
                     7.00
```

Comments

- **min**, an aggregate function, finds the minimum value in a column.
- For general information about aggregate functions, see "Aggregate Functions" on page 2-6.
- **min** can be used with numeric, character, and *datetime* columns. It cannot be used with *bit* columns. With character columns, **min** finds the lowest value in the sort sequence. **min** implicitly converts *char* datatypes to *varchar*, stripping all trailing blanks. **min** ignores null values. **distinct** is not available, since it is not meaningful with **min**.
- Adaptive Server goes directly to the first qualifying row for **min** when there is an index on the aggregated column, unless:
  - The *expression* is not a column
  - The column is not the first column of an index
  - There is another aggregate in the query
  - There is a **group by** clause

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	By default, **min** is not compliant; use **set ansinull on** for compliant behavior.

Permissions

Any user can execute **min.**

See Also

Commands	**compute Clause**, **group by and having Clauses**, **select**, **where Clause**
Functions	**avg**, **max**

mut_excl_roles

Function

Returns information about the mutual exclusivity between two roles.

Syntax

```
mut_excl_roles (role1, role2 [membership |
   activation])
```

Arguments

role1 – is one user-defined role in a mutually exclusive relationship.

role2 – is the other user-defined role in a mutually exclusive relationship.

level – is the level (membership or activation) at which the specified roles are exclusive.

Examples

```
1. alter role admin add exclusive membership
   supervisor
   select
   mut_excl_roles("admin", "supervisor", "membership")

   -----------
             1
```

Shows that the *admin* and *supervisor* roles are mutually exclusive.

Comments

- **mut_excl_roles**, a system function, returns information about the mutual exclusivity between two roles. If the System Security Officer defines *role1* as mutually exclusive with *role2* or a role directly contained by *role2*, **mut_excl_roles** returns 1; if the roles are not mutually exclusive, **mut_excl_roles** returns 0.
- For general information about system functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **mut_excl_roles.**

See Also

Commands	**alter role**, **create role**, **drop role**, **grant**, **set**, **revoke**
Functions	**proc_role**, **role_contain**, **role_id**, **role_name**
System Procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**, **sp_role**

object_id

Function

Returns the object ID of the specified object.

Syntax

object_id(*object_name*)

Arguments

object_name – is the name of a database object, such as a table, view, procedure, trigger, default, or rule. The name can be fully qualified (that is, it can include the database and owner name). Enclose the *object_name* in quotes.

Examples

```
1. select object_id("titles")

   -----------
     208003772

2. select object_id("master..sysobjects")

   -----------
             1
```

Comments

- **object_id**, a system function, returns the object's ID. Object IDs are stored in the *id* column of *sysobjects*.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **object_id**.

See Also

Functions	**col_name**, **db_id**, **object_name**
System Procedures	**sp_help**

object_name

Function

Returns the name of the object whose object ID is specified.

Syntax

object_name(*object_id*[, *database_id*])

Arguments

object_id – is the object ID of a database object, such as a table, view, procedure, trigger, default, or rule. Object IDs are stored in the *id* column of *sysobjects*.

database_id – is the ID for a database if the object is not in the current database. Database IDs are stored in the *db_id* column of *sysdatabases*.

Examples

```
1. select object_name(208003772)

   ------------------------------
   titles

2. select object_name(1, 1)

   ------------------------------
   sysobjects
```

Comments

- **object_name**, a system function, returns the object's name.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **object_name**.

See Also

Functions	**col_name**, **db_name**, **object_id**
System Procedures	**sp_help**

patindex

Function

Returns the starting position of the first occurrence of a specified pattern.

Syntax

```
patindex("%pattern%", char_expr [, using {bytes |
   characters | chars} ] )
```

Arguments

pattern – is a character expression of the *char* or *varchar* datatype that may include any of the pattern-match wildcard characters supported by Adaptive Server. The % wildcard character must precede and follow *pattern* (except when searching for first or last characters). For a description of the wildcard characters that can be used in *pattern*, see "Pattern Matching with Wildcard Characters" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

using – specifies a format for the starting position.

bytes – returns the offset in bytes.

chars or **characters** – returns the offset in characters (the default).

Examples

```
1. select au_id, patindex("%circus%", copy)
   from blurbs
```

```
au_id
 ----------- -----------
 486-29-1786           0
 648-92-1872           0
 998-72-3567          38
 899-46-2035          31
 672-71-3249           0
 409-56-7008           0
```

Selects the author ID and the starting character position of the word "circus" in the *copy* column.

```
2. select au_id, patindex("%circus%", copy,
        using chars)
   from blurbs
3. select au_id, patindex("%circus%", copy,
        using chars)
   from blurbs
```

The same as example 1.

```
4. select name
   from sysobjects
   where patindex("sys[a-d]%", name) > 0
```

```
name
------------------------------
sysalternates
sysattributes
syscharsets
syscolumns
syscomments
sysconfigures
sysconstraints
syscurconfigs
sysdatabases
sysdepends
sysdevices
```

Finds all the rows in *sysobjects* that start with "sys" and whose fourth character is "a", "b", "c", or "d".

Comments

- **patindex**, a string function, returns an integer representing the starting position of the first occurrence of *pattern* in the specified character expression, or a zero if *pattern* is not found.
- **patindex** can be used on all character data, including *tex*t and *image* data.
- By default, **patindex** returns the offset in characters; to return the offset in bytes (multibyte character strings), specify **using bytes**.
- Include percent signs before and after *pattern*. To look for *pattern* as the first characters in a column, omit the preceding %. To look for *pattern* as the last characters in a column, omit the trailing %.
- If *char_expr* is NULL, returns 0.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **patindex.**

See Also

Functions	**charindex**, **substring**

pi

Function

Returns the constant value 3.1415926535897936.

Syntax

```
pi()
```

Arguments

None.

Examples

```
1. select pi()

   -------------------
            3.141593
```

Comments

- **pi**, a mathematical function, returns the constant value of 3.1415926535897931.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **pi**.

See Also

Functions	**degrees**, **radians**

power

Function

Returns the value that results from raising the specified number to a given power.

Syntax

power(*value*, *power*)

Arguments

value – is a numeric value.

power – is an exact numeric, approximate numeric, or money value.

Examples

```
1. select power(2, 3)

   -----------
             8
```

Comments

- **power**, a mathematical function, returns the value of *value* raised to the power *power*. Results are of the same type as *value*.

  For expressions of type *numeric* or *decimal*, the results have an internal precision of 77 and a scale equal to that of the expression.

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **power**.

See Also

Functions	**exp**, **log**, **log10**

proc_role

Function

Returns information about whether the user has been granted the specified role.

Syntax

```
proc_role ("role_name")
```

Arguments

role_name – is the name of a system or user-defined role.

Examples

1.
```
create procedure sa_check as
if (proc_role("sa_role") > 0)
begin
     return(1)
end
print "You are a System Administrator."
```

   Creates a procedure to check if the user is a System Administrator.

2.
```
select proc_role("sso_role")
```

   Checks that the user has been granted the System Security Officer role.

3.
```
select proc_role("oper_role")
```

   Checks that the user has been granted the Operator role.

Comments

- **proc_role**, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- **proc_role** returns 0 if any of the following are true:
  - the user has not been granted the specified role
  - the user has not been granted a role which contains the specified role
  - the user has been granted, but has not activated, the specified role

- **proc_role** returns 1 if the invoking user has been granted, and has activated, the specified role
- **proc_role** returns 2 if the invoking user has a currently active role which contains the specified role, but the specified role has not been activated.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **proc_role.**

See Also

Commands	**alter role**, **create role**, **drop role**, **grant**, **set**, **revoke**
Functions	**mut_excl_roles**, **role_contain**, **role_id**, **role_name**, **show_role**
System Procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**, **sp_role**

ptn_data_pgs

Function

Returns the number of data pages used by a partition.

Syntax

ptn_data_pgs(*object_id*, *partition_id*)

Arguments

object_id – is the object ID for a table, stored in the *id* column of *sysobjects*, *sysindexes*, and *syspartitions*.

partition_id – is the partition number of a table.

Examples

```
1. select ptn_data_pgs(object_id("salesdetail"), 1)

   -----------
             5
```

Comments

- **ptn_data_pgs**, a system function, returns the number of data pages in a partitioned table.
- Use the **object_id** function to get an object's ID, and use **sp_helpartiton** to list the partitions in a table.
- The data pages returned by **ptn_data_pgs** may be inaccurate. Use the **update partition statistics**, **dbcc checktable**, **dbcc checkdb**, or **dbcc checkalloc** commands before using **ptn_data_pgs** to get the most accurate value.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the table owner can execute **ptn_data_pgs**.

See Also

Commands	**update partition statistics**
Functions	**data_pgs**, **object_id**
System Procedures	**sp_helpartition**

radians

Function

Returns the size, in radians, of an angle with the specified number of degrees.

Syntax

```
radians(numeric)
```

Arguments

numeric – is any exact numeric (*numeric*, *dec*, *decimal*, *tinyint*, *smallint*, or *int*), approximate numeric (*float*, *real*, or *double precision*), or *money* column, variable, constant expression, or a combination of these.

Examples

```
1. select radians(2578)

   -----------
            44
```

Comments

- **radians**, a mathematical function, converts degrees to radians. Results are of the same type as *numeric*.

  For expressions of type numeric or decimal, the results have an internal precision of 77 and a scale equal to that of the numeric expression.

  When money datatypes are used, internal conversion to *float* may cause loss of precision.

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **radians**.

See Also

Functions	**degrees**

rand

Function

Returns a random value between 0 and 1, which is generated using the specified seed value.

Syntax

rand([*integer*])

Arguments

integer – is any integer (*tinyint*, *smallint* or *int)* column name, variable, constant expression, or a combination of these.

Examples

```
1. select rand()

   --------------------
             0.395740

2. declare @seed int
   select @seed=100
   select rand(@seed)

   --------------------
               0.000783
```

Comments

- **rand**, a mathematical function, returns a random float value between 0 and 1, using the optional integer as a seed value.
- The **rand** function uses the output of a 32-bit pseudo-random integer generator. The integer is divided by the maximum 32-bit integer to give a double value between 0.0 and 1.0. The **rand** function is seeded randomly at server start-up, so getting the same sequence of random numbers is unlikely, unless the user first initializes this function with a constant seed value. The **rand** function is a global resource. Multiple users calling the **rand** function progress along a single stream of pseudo-random values. If a repeatable series of random numbers is needed, the user must assure that the function is seeded with the same value initially and that no other user calls **rand** while the repeatable sequence is desired.
- For general information about mathematical functions, see “Mathematical Functions” on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **rand**.

See Also

Datatypes	"Approximate Numeric Datatypes"

replicate

Function

Returns a string consisting of the specified expression repeated a given number of times.

Syntax

```
replicate (char_expr, integer_expr)
```

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

integer_expr – is any integer (*tinyint*, *smallint*, or *int*) column name, variable, or constant expression.

Examples

```
1. select replicate("abcd", 3)

   ------------
   abcdabcdabcd
```

Comments

- **replicate**, a string function, returns a string with the same datatype as *char_expr*, containing the same expression repeated the specified number of times or as many times as will fit into a 255-byte space, whichever is less.
- If *char_expr* is NULL, returns a single NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **replicate**.

See Also

Functions	**stuff**

reserved_pgs

Function

Returns the number of pages allocated to the specified table or index.

Syntax

reserved_pgs(*object_id*, {doampg|ioampg})

Arguments

object_id – is a numeric expression that is an object ID for a table, view, or other database object. These are stored in the *id* column of *sysobjects*.

doampg | ioampg – specifies table (**doampg**) or index (**ioampg**).

Examples

```
1. select reserved_pgs(id, doampg)
   from sysindexes where id =
       object_id("syslogs")

   -------------
            534
```

Returns the page count for the *syslogs* table.

Comments

- **reserved_pgs**, a system function, Returns the number of pages allocated to a table or an index.
- **reserved_pgs does** report pages used for internal structures.
- **reserved_pgs** works only on objects in the current database.
- For general information about system functions, see “System Functions” on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **reserved_pgs**.

See Also

Commands	**update statistics**
Functions	**data_pgs**, **used_pgs**

reverse

Function

Returns the specified string with characters listed in reverse order.

Syntax

reverse(*expression*)

Arguments

expression – is a character- or binary-type column name, variable, or constant expression of *char*, *varchar*, *nchar*, *nvarchar*, *binary*, or *varbinary* type.

Examples

```
1. select reverse("abcd")

   ----
   dcba

2. select reverse(0x12345000)

   ----------
   0x00503412
```

Comments

- **reverse**, a string function, returns the reverse of *expression*.
- If *expression* is NULL, returns NULL.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **reverse**.

See Also

Functions	**lower**, **upper**

right

Function

The rightmost part of the expression with the specified number of characters.

Syntax

right(*expression*, *integer_expr*)

Arguments

expression – is a character- or binary-type column name, variable, or constant expression of *char*, *varchar*, *nchar*, *nvarchar*, *binary*, or *varbinary* type.

integer_expr – is any integer (*tinyint*, *smallint*, or *int*) column name, variable, or constant expression.

Examples

```
1. select right("abcde", 3)

   ---
   cde

2. select right("abcde", 2)

   --
   de

3. select right("abcde", 6)

   -----
   abcde

4. select right(0x12345000, 3)

   -------
   0x345000

5. select right(0x12345000, 2)

   ------
   0x5000

6. select right(0x12345000, 6)

   ---------
   0x12345000
```

Comments

- **right**, a string function, returns the specified number of characters from the rightmost part of the character or binary expression.
- The return value has the same datatype as the character or binary expression.
- If *expression* is NULL, returns NULL.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **right**.

See Also

Functions	**rtrim**, **substring**

role_contain

Function

Returns 1 if *role2* contains *role1*.

Syntax

```
role_contain("role1", "role2")
```

Arguments

role1 – is the name of a system or user-defined role.

role2 – is the name of another system or user-defined role.

Examples

```
1. select role_contain("intern_role", "doctor_role")

   -----------
   1

2. select role_contain("specialist_role",
   "intern_role")

   -----------
   0
```

Comments

- **role_contain**, a system function, returns 1 if *role1* is contained by *role2*.
- For more information about contained roles and role hierarchies, see "Defining and Changing a Role Hierarchy" in Chapter 6, "Managing Adaptive Server Logins and Database Users" in the *System Administration Guide.*
- For more information about system functions, see "System Functions" on page 2-23

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **role_contain.**

See Also

Functions	**mut_excl_roles**, **proc_role**, **role_id**, **role_name**
Commands	**alter role**
System Procedures	**sp_activeroles**, **sp_configure**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**, **sp_role**

role_id

Function

Returns the system role ID of the role whose name you specify.

Syntax

role_id("*role_name*")

Arguments

role_name – is the name of a system or user-defined role. Role names and role IDs are stored in the *syssrvroles* system table.

Examples

```
1. select role_id("sa_role")

   ------
   0
```

Returns the system role ID of **sa_role**.

```
2. select role_id("intern_role")

   ------
   6
```

Returns the system role ID of the “intern_role”.

Comments

- **role_id**, a system function, returns the system role ID (*srid*). System role IDs are stored in the *srid* column of the *syssrvroles* system table.
- If the *role_name* is not a valid role in the system, Adaptive Server returns NULL.
- For more information about roles, see “Creating and Assigning Roles to Users” in Chapter 6, “Managing Adaptive Server Logins and Database Users” in the *System Administration Guide.*
- For more information about system functions, see “System Functions” on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **role_id.**

See Also

Functions	**mut_excl_roles**, **proc_role**, **role_contain**, **role_name**

role_name

Function

Returns the name of a role whose system role ID you specify.

Syntax

```
role_name(role_id)
```

Arguments

role_id – is the system role ID (*srid*) of the role. Role names are stored in *syssrvroles*.

Examples

```
1. select role_name(01)

   ------------------------------
   sso_role
```

Comments

- **role_name**, a system function, returns the role name.
- For more information about roles, see "Creating and Assigning Roles to Users" in Chapter 6, "Managing Adaptive Server Logins and Database Users" in the *System Administration Guide*.
- For more information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **role_name**.

See Also

Functions	mut_excl_roles, proc_role, role_contain, role_id

round

Function

Returns the value of the specified number, rounded to a given number of decimal places.

Syntax

round(*number*, *decimal_places*)

Arguments

number – is any exact numeric (*numeric*, *dec*, *decimal*, *tinyint*, *smallint*, or *int*), approximate numeric (*float*, *real*, or *double precision*), or *money* column, variable, constant expression, or a combination of these.

decimal_places – is the number of decimal places to round to.

Examples

```
1. select round(123.4545, 2)
   ----------
     123.4500
2. select round(123.45, -2)
   ----------
     100.00
3. select round(1.2345E2, 2)
   -----------------
          123.450000
4. select round(1.2345E2, -2)
   -----------------
         100.000000
```

Comments

- **round**, a mathematical function, rounds the *number* so that it has *decimal_places* significant digits.
- A positive *decimal_places* determines the number of significant digits to the right of the decimal point; a negative *decimal_places*, the number of significant digits to the left of the decimal point.
- Results are of the same type as *number* and, for numeric and decimal expressions, have an internal precision equal to the

precision of the first argument plus 1 and a scale equal to that of *number*.

- **round** always returns a value. If *decimal_places* is negative and exceeds the number of significant digits in *number*, Adaptive Server returns a result of 0. (This is expressed in the form 0.00, where the number of zeros to the right of the decimal point is equal to the scale of *numeric*.) For example:

  ```
  select round(55.55, -3)
  ```

  returns a value of 0.00.

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **round**.

See Also

Functions	**abs**, **ceiling**, **floor**, **sign**, **str**

rowcnt

Function

Returns an estimate of the number of rows in the specified table.

Syntax

```
rowcnt(sysindexes.doampg)
```

Arguments

sysindexes.doampg – is the row count maintained in *sysindexes*.

Examples

```
1. select name, rowcnt(sysindexes.doampg)
        from sysindexes
        where name in
           (select name from sysobjects
            where type = "U")
```

```
name
------------------------------ -----------
roysched                                87
salesdetail                            116
stores                                   7
discounts                                4
au_pix                                   0
blurbs                                   6
```

Comments

- **rowcnt**, a system function, returns the estimated number of rows in a table.
- The value returned by **rowcnt** can vary unexpectedly when Adaptive Server reboots and recovers transactions. The value is most accurate after running one of the following commands:
  - **dbcc checkalloc**
  - **dbcc checkdb**
  - **dbcc checktable**
  - **update all statistics**
  - **update statistics**
- For general information about system functions, see “System Functions” on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **rowcnt.**

See Also

Catalog stored procedures	**sp_statistics**
Commands	**dbcc**, **update statistics**
Functions	**data_pgs**
System procedures	**sp_helpartition**, **sp_spaceused**

rtrim

Function

Returns the specified expression, trimmed of trailing blanks.

Syntax

rtrim(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select rtrim("abcd    ")

   --------
   abcd
```

Comments

- **rtrim**, a string function, removes trailing blanks.
- If *char_expr* is NULL, returns NULL.
- Only values equivalent to the space character in the current character set are removed.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **rtrim**.

See Also

Functions	ltrim

show_role

Function

Shows the login's currently active system-defined roles.

Syntax

```
show_role()
```

Arguments

None.

Examples

```
1. select show_role()

   sa_role sso_role oper_role replication_role

2. if charindex("sa_role", show_role()) >0
   begin
        print "You have sa_role"
   end
```

Comments

- **show_role**, a system function, returns the login's current active system-defined roles, if any (**sa_role**, **sso_role**, **oper_role**, or **replication_role**). If the login has no roles, **show_role** returns NULL.
- When a Database Owner invokes **show_role** after using **setuser**, **show_role** displays the active roles of the Database Owner, not the user impersonated with **setuser**.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **show_role**.

See Also

Commands	**alter role**, **create role**, **drop role**, **grant**, **set**, **revoke**
Functions	**proc_role**, **role_contain**
System Procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**, **sp_role**

show_sec_services

Function

Lists the security services that are active for the session.

Syntax

```
show_sec_services()
```

Arguments

None.

Examples

```
1. select show_sec_services()
```

```
encryption, replay_detection
```

Shows that the user's current session is encrypting data and performing replay detection checks.

Comments

- Use **show_sec_services** to list the security services that are active during the session.
- If no security services are active, **show_sec_services** returns NULL.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **show_sec_services**.

See Also

Functions	is_sec_service_on

sign

Function

Returns the sign (+1 for positive, 0, or -1 for negative) of the specified value.

Syntax

sign(*numeric*)

Arguments

numeric – is any exact numeric (*numeric*, *dec*, *decimal*, *tinyint*, *smallint*, or *int*), approximate numeric (*float*, *real*, or *double precision*), or *money* column, variable, constant expression, or a combination of these.

Examples

```
1. select sign(-123)

   -----------
            -1
2. select sign(0)

   -----------
             0
3. select sign(123)

   -----------
             1
```

Comments

- **sign**, a mathematical function, returns the positive (+1), zero (0), or negative (-1).
- Results are of the same type, and have the same precision and scale, as the numeric expression.
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **sign.**

See Also

Functions	**abs**, **ceiling**, **floor**, **round**

sin

Function

Returns the sine of the specified angle (in radians).

Syntax

sin(*approx_numeric*)

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

Examples

```
1. select sin(45)

   --------------------
             0.850904
```

Comments

- **sin**, a mathematical function, returns the sine of the specified angle (measured in radians).
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **sin**.

See Also

Functions	**cos**, **degrees**, **radians**

sortkey

Function

Generates values that can be used to order results based on collation behavior, which allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity.

Syntax

```
sortkey (char_expression [, {collation_name |
   collation_ID}])
```

Arguments

char_expression is one of the following:

- Character type (*char, varchar, nchar,* or *nvarchar*)
- Character variable, or
- Constant character expression, enclosed in single or double quotation marks

collation_name is a quoted string or a character variable that specifies the collation to use.

collation_ID is an integer constant or a variable that specifies the collation to use.

Comments

- The **sortkey** function generates values that can be used to order results based on collation behavior. This allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity. The return value is a *varbinary* datatype value that contains coded collation information for the input string that is retained from the **sortkey** function.

  For example, you can store the values returned by **sortkey** in a column with the source character string. When you want to retrieve the character data in the desired order, the **select** statement only needs to include an **order by** clause on the columns that contain the results of running **sortkey.**

The **sortkey** function guarantees that the values it returns for a given set of collation criteria work for the binary comparisons that are performed on *varbinary* datatypes.

- *char_expression* must be composed of characters that are encoded in the server's default character set.
- *char_expression* can be an empty string. If it is an empty string:
  - **sortkey** returns a zero-length *varbinary* value, and
  - Adaptive Server stores a blank for the empty string.

  An empty string has a different collation value than an NULL string from a database column.
- If *char_expression* is NULL, **sortkey** returns a NULL value.
- If you do not specify a value for *collation_name* or *collation_ID*, **sortkey** assumes binary collation.

➤ ***Note***

sortkey can generate up to 6 bytes of collation information for each input character. Therefore, the result from using **sortkey** may exceed the 255-byte length limit of the *varbinary* datatype. If this happens, the result is truncated to fit. Truncation removes result bytes for each input character until the result string is less than 255 bytes. If this occurs, a warning message is issued, but the query or transaction that contained the **sortkey** function continues to work.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **sortkey.**

See Also

Functions	compare

soundex

Function

Returns a 4-character code representing the way an expression sounds.

Syntax

soundex(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select soundex ("smith"), soundex ("smythe")

   ----- -----
   S530  S530
```

Comments

- **soundex**, a string function, returns a 4-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte roman letters.
- The **soundex** function converts an alpha string to a four-digit code for use in locating similar-sounding words or names. All vowels are ignored unless they constitute the first letter of the string.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **soundex**.

See Also

Functions	**difference**

space

Function

Returns a string consisting of the specified number of single-byte spaces.

Syntax

space(*integer_expr*)

Arguments

integer_expr – is any integer (*tinyint*, *smallint*, or *int*) column name, variable, or constant expression.

Examples

```
1. select "aaa", space(4), "bbb"

   --- ---- ---
   aaa      bbb
```

Comments

- **space**, a string function, returns a string with the indicated number of single-byte spaces.
- For general information about string functions, see "String Functions" on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **space**.

See Also

Functions	**isnull**, **isnull**, **rtrim**

sqrt

Function

Returns the square root of the specified number.

Syntax

sqrt(*approx_numeric*)

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression that evaluates to a positive number.

Examples

```
1. select sqrt(4)

    2.000000
```

Comments

- **sqrt**, a mathematical function, returns the square root of the specified value.
- If you attempt to select the square root of a negative number, Adaptive Server returns the following error message:

  ```
  Domain error occurred.
  ```

- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **sqrt**.

See Also

Functions	power

str

Function

Returns the character equivalent of the specified number.

Syntax

str(*approx_numeric* [, *length* [, *decimal*]])

Arguments

approx_numeric – is any approximate numeric (*float*, *real*, or *double precision*) column name, variable, or constant expression.

length – sets the number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.

decimal – sets the number of decimal digits to be returned. The default is 0.

Examples

```
1. select str(1234.7, 4)

   ----
   1235

2. select str(-12345, 6)

   ------
   -12345

3. select str(123.45, 5, 2)

   -----
   123.5
```

Comments

- **str**, a string function, returns a character representation of the floating point number. For general information about string functions, see “String Functions” on page 2-22.
- *length* and *decimal* are optional. If given, they must be nonnegative. **str** rounds the decimal portion of the number so that the results fit within the specified length. The length should be long enough to accommodate the decimal point and, if negative, the number’s sign. The decimal portion of the result is rounded to fit within the specified length. If the integer portion of the number

does not fit within the length, however, **str** returns a row of asterisks of the specified length. For example:

```
select str(123.456, 2, 4)

--
**
```

A short *approx_numeric* is right justified in the specified length, and a long *approx_numeric* is truncated to the specified number of decimal places.

- If *approx_numeric* is NULL, returns NULL.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **str.**

See Also

Functions	**abs**, **ceiling**, **floor**, **round**, **sign**

stuff

Function

Returns the string formed by deleting a specified number of characters from one string and replacing them with another string.

Syntax

stuff(*char_expr1, start, length, char_expr2*)

Arguments

char_expr1 – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

start – specifies the character position at which to begin deleting characters.

length – specifies the number of characters to delete.

char_expr2 – is another character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select stuff("abc", 2, 3, "xyz")

   ----
   axyz

2. select stuff("abcdef", 2, 3, null)

   go
   ---
   aef

3. select stuff("abcdef", 2, 3, "")

   ----
   a ef
```

Comments

- **stuff**, a string function, deletes *length* characters from *char_expr1* at *start*, then inserts *char_expr2* into *char_expr1* at *start*. For general information about string functions, see “String Functions” on page 2-22.
- If the start position or the length is negative, a NULL string is returned. If the start position is longer than *expr1*, a NULL string

is returned. If the length to be deleted is longer than *expr1*, *expr1* is deleted through its last character (see example 1).

- To use **stuff** to delete a character, replace *expr2* with "NULL" rather than with empty quotation marks. Using " " to specify a null character replaces it with a space (see examples 2 and 3).
- If *char_expr1* is NULL, returns NULL. If *char_expr1* is a string value and *char_expr2* is NULL, replaces the deleted characters with nothing.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **stuff**.

See Also

Functions	**replicate**, **substring**

substring

Function

Returns the string formed by extracting the specified number of characters from another string.

Syntax

substring(*expression, start, length*)

Arguments

expression – is a binary or character column name, variable or constant expression. Can be *char*, *varchar*, *nchar* or *nvarchar* data, *binary* or *varbinary*.

start – specifies the character position at which the substring begins.

length – specifies the number of characters in the substring.

Examples

1. `select au_lname, substring(au_fname, 1, 1) from authors`

   Displays the last name and first initial of each author, for example, "Bennet A."

2. `select substring(upper(au_lname), 1, 3) from authors`

   Converts the author's last name to uppercase, then displays the first three characters.

3. `select substring((pub_id + title_id), 1, 6)`

   Concatenates *pub_id* and *title_id*, then displays the first six characters of the resulting string.

4. **`select substring(xactid,5,2) from syslogs`**

   Extracts the lower four digits from a binary field, where each position represents two binary digits:

Comments

- **substring**, a string function, returns part of a character or binary string. For general information about string functions, see "String Functions" on page 2-22.
- If any of the arguments to **substring** are NULL, returns NULL.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **substring.**

See Also

Functions	**charindex**, **patindex**, **stuff**

sum

Function

Returns the total of the values.

Syntax

```
sum([all | distinct] expression)
```

Arguments

all – applies **sum** to all values. **all** is the default.

distinct – eliminates duplicate values before **sum** is applied. **distinct** is optional.

expression – is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

1.
```
select avg(advance), sum(total_sales)
from titles
where type = "business"
```

   Calculates the average advance and the sum of total sales for all business books. Each of these aggregate functions produces a single summary value for all of the retrieved rows.

2.
```
select type, avg(advance), sum(total_sales)
from titles
group by type
```

   Used with a **group by** clause, the aggregate functions produce single values for each group, rather than for the whole table. This statement produces summary values for each type of book.

3.
```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > $25000 and avg(price) > $15
```

   Groups the *titles* table by publishers, and includes only those groups of publishers who have paid more than $25,000 in total advances and whose books average more than $15 in price.

Comments

- **sum**, an aggregate function, finds the sum of all the values in a column. **sum** can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating sums.
- For general information about aggregate functions, see "Aggregate Functions" on page 2-6.
- When you sum integer data, Adaptive Server treats the result as an *int* value, even if the datatype of the column is *smallint* or *tinyint*. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type *int*.
- You cannot use **sum** with the binary datatypes.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Transact-SQL extension	By default, **sum** is not compliant. **set ansinull on** for compliant behavior.

Permissions

Any user can execute **sum**.

See Also

Commands	**compute Clause**, **group by and having Clauses**, **select**, **where Clause**
Functions	**count**, **max**, **min**

suser_id

Function

Returns the server user's ID number from the *syslogins* table.

Syntax

suser_id([*server_user_name*])

Arguments

server_user_name – is an Adaptive Server login name.

Examples

```
1. select suser_id()

   ------
        1

2. select suser_id("margaret")

   ------
        5
```

Comments

- **suser_id**, a system function, returns the server user's ID number from *syslogins*. For general information about system functions, see "System Functions" on page 2-23.
- To find the user's ID in a specific database from the *sysusers* table, use the **user_id** system function.
- If no *server_user_name* is supplied, **suser_id** returns the server ID of the current user.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **suser_id**.

See Also

Functions	**suser_name**, **user_id**

suser_name

Function

Returns the name of the current server user or the user whose server ID is specified.

Syntax

suser_name([*server_user_id*])

Arguments

server_user_name – is an Adaptive Server user ID.

Examples

```
1. select suser_name()
   ------------------------------
   sa
2. select suser_name(4)
   ------------------------------
   margaret
```

Comments

- **suser_name**, a system function, returns the server user's name. Server user IDs are stored in *syslogins*. If no *server_user_id* is supplied, **suser_name** returns the name of the current user.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **suser_name**.

See Also

Functions	**suser_id**, **user_name**

syb_sendmsg

Function

Sends a message to a UDP (User Datagram Protocol) port.

Syntax

syb_sendmsg *ip_address*, *port_number*, *message*

Arguments

ip_address – is the IP address of the machine where the UDP application is running.

port_number – is the port number of the UDP port.

Examples

1. **select syb_sendmsg("120.10.20.5", 3456, "Hello")**

   Sends the message “Hello” to port 3456 at IP address 120.10.20.5.

2. ```
   declare @msg varchar(255)
   select @msg = "Message to send"
   select syb_sendmsg (ip_address, portnum, @msg)
   from sendports
   where username = user_name()
   ```

   Reads the IP address and port number from a user table, and uses a variable for the message to be sent.

Comments

- **sp_sendmsg** is not supported on Windows NT.
- A System Security Officer must set the configuration parameter **allow sendmsg** to 1 to enable UDP messaging.
- There are no security checks with **sp_sendmsg.** Sybase strongly recommends caution when using **sp_sendmsg** to send sensitive information across the network. By enabling this functionality, the user accepts any security problems which result from its use.
- For more a sample C program that creates a UDP port, see the **sp_sendmsg.**
   ```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **syb_sendmsg.**

See Also

System Procedures	sp_sendmsg

tan

Function

Returns the tangent of the specified angle (in radians).

Syntax

```
tan(angle)
```

Arguments

angle – is the size of the angle in radians, expressed as a column name, variable, or expression of type *float*, *real*, *double precision*, or any datatype that can be implicitly converted to one of these types.

Examples

```
1. select tan(60)

   --------------------
              0.320040
```

Comments

- **tan**, a mathematical function, returns the tangent of the specified angle (measured in radians).
- For general information about mathematical functions, see "Mathematical Functions" on page 2-20.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **tan**.

See Also

Functions	**atan**, **atn2**, **degrees**, **radians**

textptr

Function

Returns the 16-byte *varbinary* pointer to the first page of the specified *text* or *image* column.

Syntax

textptr(*column_name*)

Arguments

column_name – is the name of a *text* column.

Examples

1.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
where au_id = “486-29-1786”
readtext blurbs.copy @val 1 5
```

   This example uses the **textptr** function to locate the *text* column, *copy*, associated with *au_id* 486-29-1786 in the author's *blurbs* table. The text pointer is put into a local variable *@val* and supplied as a parameter to the **readtext** command, which returns 5 bytes, starting at the second byte (offset of 1).

2.
```
select au_id, textptr(copy) from blurbs
```

   Selects the *title_id* column and the 16-byte text pointer of the *blurb* column from the *texttest* table.

Comments

- **textptr**, a text and image function, returns the text pointer value, a 16-byte *varbinary* value. The text pointer is checked to ensure that it points to the first text or image page.
- If a *text* or an *image* column has not been initialized by a non-null **insert** or by any **update** statement, **textptr** returns a NULL pointer. Use **textvalid** to check whether a text pointer exists. You cannot use **writetext** or **readtext** without a valid text pointer.
- For general information about text and image functions, see “Text and Image Functions” on page 2-25.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **textptr.**

See Also

Datatypes	"text and image Datatypes"
Functions	**textvalid**

textvalid

Function

Returns 1 if the pointer to the specified *text* column is valid; 0 if it is not.

Syntax

```
textvalid("table_name.column_name", textpointer)
```

Arguments

"*table_name.column_name*" – is the name of a table and its *text* column.

textpointer – is a text pointer value.

Examples

```
1. select textvalid ("texttest.blurb",
   textptr(blurb)) from texttest
```

Reports whether a valid text pointer exists for each value in the *blurb* column of the *texttest* table.

Comments

- **textvalid**, a text and image function, checks that a given text pointer is valid. Returns 1 if the pointer is valid or 0 if it is not.
- The identifier for a *text* or an *image* column must include the table name.
- For general information about text and image functions, see "Text and Image Functions" on page 2-25.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **textvalid**.

See Also

Datatypes	"text and image Datatypes"
Functions	**textptr**

tsequal

Function

Compares *timestamp* values to prevent update on a row that has been modified since it was selected for browsing.

Syntax

tsequal(*browsed_row_timestamp, stored_row_timestamp*)

Arguments

browsed_row_timestamp – is the *timestamp* column of the browsed row.

stored_row_timestamp – is the *timestamp* column of the stored row.

Examples

```
1. update publishers
   set city == "Springfield"
   where pub_id = "0736"
   and tsequal(timestamp, 0x0001000000002ea8)
```

Retrieves the *timestamp* column from the current version of the *publishers* table and compares it to the value in the *timestamp* column that has been saved. If the values in the two *timestamp* columns are equal, updates the row. If the values are not equal, returns an error message.

Comments

- **tsequal**, a system function, compares the *timestamp* column values to prevent an update on a row that has been modified since it was selected for browsing. For general information about system functions, see “System Functions” on page 2-23.
- **tsequal** allows you to use browse mode without calling the **dbqual** function in DB-Library. Browse mode supports the ability to perform updates while viewing data. It is used in front-end applications using Open Client and a host programming language. A table can be browsed if its rows have been timestamped.
- To browse a table in a front-end application, append the **for browse** keywords to the end of the **select** statement sent to Adaptive Server.

  For example:

Start of **select** *statement in an Open Client application*

```
...
   for browse
```

Completion of the Open Client application routine

- The **tsequal** function should not be used in the **where** clause of a select statement, only in the where clause of **insert** and **update** statements where the rest of the **where** clause matches a single unique row.

  If a *timestamp* column is used as a search clause, it should be compared like a regular *varbinary* column; that is, *timestamp1 = timestamp2*.

Timestamping a New Table for Browsing

- When creating a new table for browsing, include a column named *timestamp* in the table definition. The column is automatically assigned a datatype of *timestamp*; you do not have to specify its datatype. For example:

  ```
  create table newtable(col1 int, timestamp,
      col3 char(7))
  ```

  Whenever you insert or update a row, Adaptive Server timestamps it by automatically assigning a unique *varbinary* value to the *timestamp* column.

Timestamping an Existing Table

- To prepare an existing table for browsing, add a column named *timestamp* with **alter table**. For example:

  ```
  alter table oldtable add timestamp
  ```

  adds a *timestamp* column with a NULL value to each existing row. To generate a timestamp, update each existing row without specifying new column values. For example:

  ```
  update oldtable
  set col1 = col1
  ```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **tsequal**.

See Also

Datatypes	"Timestamp Datatype"

upper

Function

Returns the uppercase equivalent of the specified string.

Syntax

upper(*char_expr*)

Arguments

char_expr – is a character-type column name, variable, or constant expression of *char*, *varchar*, *nchar* or *nvarchar* type.

Examples

```
1. select upper("abcd")

   ----
   ABCD
```

Comments

- **upper**, a string function, converts lowercase to uppercase, returning a character value.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see “String Functions” on page 2-22.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **upper**.

See Also

Functions	lower

used_pgs

Function

Returns the number of pages used by the specified table and its clustered index, or the number of pages in a nonclustered index.

Syntax

used_pgs(*object_id, doampg, ioampg*)

Arguments

object_id – is the object ID of a table or the object ID of a table to which the index belongs.

doampg – is the page number for the object allocation map of a table or clustered index, stored in the *doampg* column of *sysindexes*.

ioampg – is the page number for the allocation map of a nonclustered index, stored in the *ioampg* column of *sysindexes*.

Examples

1.
```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("titles")

name          id          indid  doampg   ioampg
------------- ----------- ------ -------- -------
titleidind      208003772      1      560     552
titleind        208003772      2        0     456

select used_pgs(208003772, 560, 552)

-----------
          6
```

   Returns the number of pages used by the data and clustered index of the *titles* table.

2.
```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("stores")

name          id          indid  doampg   ioampg
------------- ----------- ------ -------- -------
stores          240003886      0     464       0

select used_pgs(240003886, 464, 0)

-----------
          2
```

Returns the number of pages used by the *stores* table, which has no index.

Comments

- **used_pgs**, a system function, returns the total number of pages used by a table and its clustered index, or the number of pages in a nonclustered index.
- In the examples, *indid* 0 indicates a table; *indid* 1 indicates a clustered index; an *indid* of 2–250 is a nonclustered index; and an *indid* of 255 is *text* or *image* data.
- **used_pgs** only works on objects in the current database.
- Each table and each index on a table has an object allocation map (OAM), which contains information about the number of pages allocated to and used by an object. This information is updated by most Adaptive Server processes when pages are allocated or deallocated. The **sp_spaceused** system procedure reads these values to provide quick space estimates. Some **dbcc** commands update these values while they perform consistency checks.
- For general information about system functions, see “System Functions” on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **used_pgs**.

See Also

Functions	**data_pgs**, **object_id**

user

Function

Returns the name of the current user.

Syntax

```
user
```

Arguments

None.

Examples

```
1. select user

   ------
   dbo
```

Comments

- **user**, a system function, returns the user's name.
- If the **sa_role** is active, you are automatically the Database Owner in any database you are using. Inside a database, the user name of the Database Owner is always "dbo".
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Entry level compliant

Permissions

Any user can execute **user**.

See Also

Functions	user_name

user_id

Function

Returns the ID number of the specified user or of the current user in the database.

Syntax

user_id([*user_name*])

Arguments

user_name – is the name of the user.

Examples

```
1. select user_id()

   ------
        1

2. select user_id("margaret")

   ------
        4
```

Comments

- **user_id**, a system function, returns the user's ID number. For general information about system functions, see "System Functions" on page 2-23.
- **user_id** reports the number from *sysusers* in the current database. If no *user_name* is supplied, **user_id** returns the ID of the current user. To find the server user ID, which is the same number in every database on Adaptive Server, use **suser_id**.
- Inside a database, the "guest" user ID is always 2.
- Inside a database, the **user_id** of the Database Owner is always 1. If you have the **sa_role** active, you are automatically the Database Owner in any database you are using. To return to your actual user ID, use **set sa_role off** before executing **user_id**. If you are not a valid user in the database, Adaptive Server returns an error when you use **set sa_role off**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must System Administrator or System Security Officer to use this function on a *user_name* other than your own.

See Also

Commands	**setuser**
Functions	**suser_id**, **user_name**

user_name

Function

Returns the name within the database of the specified user or of the current user.

Syntax

user_name([*user_id*])

Arguments

user_id – is the ID of a user.

Examples

```
1. select user_name()

   ------------------------------
   dbo
2. select user_name(4)

   ------------------------------
   margaret
```

Comments

- **user_name**, a system function, returns the user's name, based on the user's ID in the current database. For general information about system functions, see "System Functions" on page 2-23.
- If no *user_id* is supplied, **user_name** returns the name of the current user.
- If the **sa_role** is active, you are automatically the Database Owner in any database you are using. Inside a database, the **user_name** of the Database Owner is always "dbo".

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Administrator or System Security Officer to use this function on a *user_id* other than your own.

See Also

Functions	**suser_name**, **user_id**

valid_name

Function

Returns 0 if the specified string is not a valid identifier or a number other than 0 if the string is a valid identifier.

Syntax

valid_name(*character_expression*)

Arguments

character_expression – is a character-type column name, variable, or constant expression of *char, varchar, nchar* or *nvarchar* type. Constant expressions must be enclosed in quotation marks.

Examples

```
1. create procedure chkname
   @name varchar(30)
   as
        if valid_name(@name) = 0
        print "name not valid"
```

Creates a procedure to verify that identifiers are valid.

Comments

- **valid_name**, a system function, returns 0 if the *character_ expression* is not a valid identifier (illegal characters, more than 30 bytes long, or a reserved word), or a number other than 0 if it is a valid identifier.
- Adaptive Server identifiers can be a maximum of 30 bytes in length, whether single-byte or multibyte characters are used. The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_) character. Temporary table names, which begin with the pound sign (#), and local variable names, which begin with the at sign (@), are exceptions to this rule. **valid_name** returns 0 for identifiers that begin with the pound sign (#) and the at sign (@).
- For general information about system functions, see “System Functions” on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **valid_name.**

See Also

System Procedures	sp_checkreswords

valid_user

Function

Returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

Syntax

valid_user(*server_user_id*)

Arguments

server_user_id – is a server user ID. Server user IDs are stored in the *suid* column of *syslogins*.

Examples

```
1. select valid_user(4)

   ---------------
                 1
```

Comments

- **valid_user**, a system function, returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.
- For general information about system functions, see "System Functions" on page 2-23.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Administrator or a System Security Officer to use this function on a *server_user_id* other than your own.

See Also

System Procedures	sp_addlogin, sp_adduser

3 Expressions, Identifiers, and Wildcard Characters

This chapter describes Transact-SQL expressions, valid identifiers, and wildcard characters.

Expressions

An expression is a combination of one or more constants, literals, functions, column identifiers and / or variables, separated by operators, that returns a single value. Expressions can be of several types, including **arithmetic**, **relational**, **logical** (or **Boolean**), and **character string**. In some Transact-SQL clauses, a subquery can be used in an expression. A **case** expression can be used in an expression.

Table 3-1 lists the types of expressions that are used in Adaptive Server syntax statements.

Table 3-1: Types of expressions used in syntax statements

Usage	Definition
expression	Can include constants, literals, functions, column identifiers, variables, or parameters
logical expression	An expression that returns TRUE, FALSE, or UNKNOWN
constant expression	An expression that always returns the same value, such as “5+3” or “ABCDE”
float_expr	Any floating-point expression or an expression that implicitly converts to a floating value
integer_expr	Any integer expression or an expression that implicitly converts to an integer value
numeric_expr	Any numeric expression that returns a single value
char_expr	Any expression that returns a single character-type value
binary_expression	An expression that returns a single *binary* or *varbinary* value

Arithmetic and Character Expressions

The general pattern for arithmetic and character expressions is:

```
{constant | column_name | function | (subquery)
   | (case_expression)}
      [{arithmetic_operator | bitwise_operator |
       string_operator | comparison_operator }
   {constant | column_name | function | (subquery)
   | case_expression}]...
```

Relational and Logical Expressions

A logical expression or relational expression returns TRUE, FALSE, or UNKNOWN. The general patterns are:

```
expression comparison_operator [any | all] expression

expression [not] in expression

[not]exists expression

expression [not] between expression and expression

expression [not] like "match_string"
   [escape "escape_character"]

not expression like "match_string"
   [escape "escape_character"]

expression is [not] null

not logical_expression

logical_expression {and | or} logical_expression
```

Operator Precedence

Operators have the following precedence levels, where 1 is the highest level and 6 is the lowest:

1. unary (single argument) - + ~
2. * / %
3. binary (two argument) + - & | ^
4. **not**
5. **and**
6. **or**

When all operators in an expression are at the same level, the order of execution is left to right. You can change the order of execution with parentheses—the most deeply nested expression is processed first.

Arithmetic Operators

Adaptive Server uses the following arithmetic operators:

Table 3-2: Arithmetic operators

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulo (Transact-SQL extension)

Addition, subtraction, division, and multiplication can be used on exact numeric, approximate numeric, and money type columns.

The modulo operator cannot be used on *smallmoney*, *money*, *float* or *real* columns. Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

When you perform arithmetic operations on mixed datatypes, for example *float* and *int*, Adaptive Server follows specific rules for determining the type of the result. For more information, see Chapter 1, “System and User-Defined Datatypes.”

Bitwise Operators

The bitwise operators are a Transact-SQL extension for use with integer type data. These operators convert each integer operand into its binary representation, then evaluate the operands column by column. A value of 1 corresponds to true; a value of 0 corresponds to false.

Table 3-3 summarizes the results for operands of 0 and 1. If either operand is NULL, the bitwise operator returns NULL:

Table 3-3: Truth tables for bitwise operations

& (and)	1	0
1	1	0
0	0	0

Table 3-3: Truth tables for bitwise operations (continued)

| (or)	1	0
1	1	1
0	1	0

^ (exclusive or)	1	0
1	0	1
0	1	0

~ (not)	
1	FALSE
0	0

The examples in Table 3-4 use two *tinyint* arguments, A = 170 (10101010 in binary form) and B = 75 (01001011 in binary form).

Table 3-4: Examples of bitwise operations

Operation	Binary Form	Result	Explanation
(A & B)	10101010 01001011 ---------- 00001010	10	Result column equals 1 if both A and B are 1. Otherwise, result column equals 0.
(A | B)	10101010 01001011 ---------- 11101011	235	Result column equals 1 if either A or B, or both, is 1. Otherwise, result column equals 0
(A ^ B)	10101010 01001011 ---------- 11100001	225	Result column equals 1 if either A or B, but not both, is 1
(~A)	10101010 ---------- 01010101	85	All 1's are changed to 0's and all 0's to 1's

The String Concatenation Operator

The string operator **+** can be used to concatenate two or more character or binary expressions. For example:

1. `select Name = (au_lname + ", " + au_fname)`
   `from authors`

   Displays author names under the column heading *Name* in last-name first-name order, with a comma after the last name; for example, "Bennett, Abraham."

2. `select "abc" + "" + "def"`

   Returns the string "abc def". The empty string is interpreted as a single space in all *char*, *varchar*, *nchar*, *nvarchar*, and *text* concatenation, and in *varchar* insert and assignment statements.

When concatenating non-character, non-binary expressions, always use **convert**:

```
select "The date is " +
    convert(varchar(12), getdate())
```

A string concatenated with NULL evaluates to the value of the string. This is an exception to the SQL standard, which states that a string concatenated with a NULL should evaluate to NULL.

The Comparison Operators

Adaptive Server uses the comparison operators listed in Table 3-5:

Table 3-5: Comparison operators

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
!=	Not equal to (Transact-SQL extension)
!>	Not greater than (Transact-SQL extension)
!<	Not less than (Transact-SQL extension)

In comparing character data, **<** means closer to the beginning of the server's sort order and **>** means closer to the end of the sort order. Uppercase and lowercase letters are equal in a case-insensitive sort order. Use **sp_helpsort** to see the sort order for your Adaptive Server.

Trailing blanks are ignored for comparison purposes. So, for example, “Dirk” is the same as “Dirk ”.

In comparing dates, **<** means earlier and **>** means later.

Put single or double quotes around all character and *datetime* data used with a comparison operator:

```
= "Bennet"
> "May 22 1947"
```

Nonstandard Operators

The following operators are Transact-SQL extensions:

- Modulo operator: **%**
- Negative comparison operators: **!>**, **!<**, **!=**
- Bitwise operators: **~**, **^**, **|**, **&**
- Join operators: ***=** and **=***

Using *any*, *all* and *in*

any is used with **<**, **>**, or **=** and a subquery. It returns results when any value retrieved in the subquery matches the value in the **where** or **having** clause of the outer statement. For more information, see the *Transact-SQL User’s Guide*.

all is used with **<** or **>** and a subquery. It returns results when all values retrieved in the subquery are less than (**<**) or greater than (**>**) the value in the **where** or **having** clause of the outer statement. For more information, see the *Transact-SQL User’s Guide*.

in returns results when any value returned by the second expression matches the value in the first expression. The second expression must be a subquery or a list of values enclosed in parentheses. **in** is equivalent to **= any**. See “where Clause” for details.

Negating and Testing

not negates the meaning of a keyword or logical expression.

Use **exists**, followed by a subquery, to test for the existence of a particular result.

Ranges

between is the range-start keyword; **and** is the range-end keyword. The range:

```
where column1 between x and y
```

is inclusive.

The range:

```
where column1 > x and column1 < y
```

is not inclusive.

Using Nulls in Expressions

Use **is null** or **is not null** in queries on columns defined to allow null values.

An expression with a bitwise or arithmetic operator evaluates to NULL if any of the operands are null. For example:

```
1 + column1
```

evaluates to NULL if *column1* is NULL.

Comparisons That Return TRUE

In general, the result of comparing null values is UNKNOWN, since it is not possible to determine whether NULL is equal (or not equal) to a given value or to another NULL. However, the following cases return TRUE when *expression* is any column, variable or literal, or combination of these, which evaluates as NULL:

- *expression* **is null**
- *expression* = **null**
- *expression* = *@x*, where *@x* is a variable or parameter containing NULL. This exception facilitates writing stored procedures with null default parameters.
- *expression* != *n*, where *n* is a literal that does not contain NULL, and *expression* evaluates to NULL.

The negative versions of these expressions return TRUE when the expression does not evaluate to NULL:

- *expression* **is not null**
- *expression* != **null**

- *expression* != *@x*

Note that the far right side of these exceptions is a literal null, or a variable or parameter containing NULL. If the far right side of the comparison is an expression (such as *@nullvar* + 1), the entire expression evaluates to NULL.

Following these rules, null column values do not join with other null column values. Comparing null column values to other null column values in a **where** clause always returns UNKNOWN for null values, regardless of the comparison operator, and the rows are not included in the results. For example, this query returns no result rows where column1 contains NULL in both tables (although it may return other rows):

```
select column1
from table1, table2
where table1.column1 = table2.column1
```

Difference Between FALSE and UNKNOWN

Although neither FALSE nor UNKNOWN returns values, there is an important logical difference between FALSE and UNKNOWN, because the opposite of false ("not false") is true. For example, "1 = 2" evaluates to false and its opposite, "1 != 2", evaluates to true. But "not unknown" is still unknown. If null values are included in a comparison, you cannot negate the expression to get the opposite set of rows or the opposite truth value.

Using "NULL" As a Character String

Only columns for which NULL was specified in the **create table** statement and into which you have explicitly entered NULL (no quotes), or into which no data has been entered, contain null values. Avoid entering the character string "NULL" (with quotes) as data for a character column. It can only lead to confusion. Use "N/A", "none", or a similar value instead. When you want to enter the value NULL explicitly, do **not** use single or double quotes.

NULLs Compared to the Empty String

The empty string (" "or ' ') is always stored as a single space in variables and column data. This concatenation statement:

```
"abc" + "" + "def"
```

is equivalent to “abc def”, not to “abcdef”. The empty string is never evaluated as NULL.

Connecting Expressions

and connects two expressions and returns results when both are true. **or** connects two or more conditions and returns results when either of the conditions is true.

When more than one logical operator is used in a statement, **and** is evaluated before **or**. You can change the order of execution with parentheses.

Table 3-6 shows the results of logical operations, including those that involve null values:

Table 3-6: Truth tables for logical expressions

and	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
NULL	UNKNOWN	FALSE	UNKNOWN

or	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
NULL	TRUE	UNKNOWN	UNKNOWN

not	
TRUE	FALSE
FALSE	TRUE
NULL	UNKNOWN

The result UNKNOWN indicates that one or more of the expressions evaluates to NULL, and that the result of the operation cannot be determined to be either TRUE or FALSE. See “Using Nulls in Expressions” on page 3-7 for more information.

Using Parentheses in Expressions

Parentheses can be used to group the elements in an expression. When "expression" is given as a variable in a syntax statement, a simple expression is assumed. "Logical expression" is specified when only a logical expression is acceptable.

Comparing Character Expressions

Character constant expressions are treated as *varchar*. If they are compared with non-*varchar* variables or column data, the datatype precedence rules are used in the comparison (that is, the datatype with lower precedence is converted to the datatype with higher precedence). If implicit datatype conversion is not supported, you must use the **convert** function.

Comparison of a *char* expression to a *varchar* expression follows the datatype precedence rule; the "lower" datatype is converted to the "higher" datatype. All *varchar* expressions are converted to *char* (that is, trailing blanks are appended) for the comparison.

Using the Empty String

The empty string ("") or ('') is interpreted as a single blank in **insert** or assignment statements on *varchar* data. In concatenation of *varchar*, *char*, *nchar*, *nvarchar* data, the empty string is interpreted as a single space; for example:

```
"abc" + "" + "def"
```

is stored as "abc def". The empty string is never evaluated as NULL.

Including Quotation Marks in Character Expressions

There are two ways to specify literal quotes within a *char* or *varchar* entry. The first method is to double the quotes. For example, if you begin a character entry with a single quote and you want to include a single quote as part of the entry, use two single quotes:

```
'I don''t understand.'
```

With double quotes:

```
"He said, ""It's not really confusing."""
```

The second method is to enclose a quote in the opposite kind of quote mark. In other words, surround an entry containing a double quote with single quotes (or vice versa). Here are some examples:

```
'George said, "There must be a better way."'
"Isn't there a better way?"
'George asked, "Isn"t there a better way?"'
```

Using the Continuation Character

To continue a character string to the next line on your screen, enter a backslash (\) before going to the next line.

Identifiers

Identifiers are names for database objects such as databases, tables, views, columns, indexes, triggers, procedures, defaults, rules, and cursors.

Adaptive Server identifiers can be a maximum of 30 bytes in length, whether single-byte or multibyte characters are used. The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_) character.

➤ ***Note***

Temporary table names, which begin with the pound sign (#), and local variable names, which begin with the at sign(@), are exceptions to this rule.

Subsequent characters can include letters, numbers, the symbols #, @, _, and currency symbols such as $ (dollars), ¥ (yen), and £ (pound sterling). Identifiers cannot include special characters such as !, %, ^, &, *, and . or embedded spaces.

You cannot use a reserved word, such as a Transact-SQL command, as an identifier. For a complete list of reserved words, see Chapter 4, "Reserved Words."

Tables Beginning with # (Temporary Tables)

Tables whose names begin with the pound sign (#) are temporary tables. You cannot create other types of objects whose names begin with the pound sign.

Adaptive Server performs special operations on temporary table names to maintain unique naming on a per-session basis. Long temporary table names are truncated to 13 characters (including the pound sign); short names are padded to 13 characters with underscores (_). A 17-digit numeric suffix that is unique for an Adaptive Server session is appended.

Case Sensitivity and Identifiers

Sensitivity to the case (upper or lower) of identifiers and data depends on the sort order installed on your Adaptive Server. Case sensitivity can be changed for single-byte character sets by reconfiguring Adaptive Server's sort order (see the *System Administration Guide* for more information). Case is significant in utility program options.

If Adaptive Server is installed with a case-insensitive sort order, you cannot create a table named *MYTABLE* if a table named MyTable or mytable already exists. Similarly, this command:

```
select * from MYTABLE
```

will return rows from *MYTABLE*, MyTable, or mytable, or any combination of uppercase and lowercase letters in the name.

Uniqueness of Object Names

Object names need not be unique in a database. However, column names and index names must be unique within a table, and other object names must be unique for each **owner** within a **database**. Database names must be unique on Adaptive Server.

Using Delimited Identifiers

Delimited identifiers are object names enclosed in double quotes. Using delimited identifiers allows you to avoid certain restrictions on object names. Table, view, and column names can be delimited by quotes; other object names cannot.

Delimited identifiers can be reserved words, can begin with non-alphabetic characters, and can include characters that would not otherwise be allowed. They cannot exceed 28 bytes.

◆ ***WARNING!***

Delimited identifiers may not be recognized by all front-end applications and should not be used as parameters to system procedures.

Before creating or referencing a delimited identifier, you must execute:

```
set quoted_identifier on
```

Each time you use the delimited identifier in a statement, you must enclose it in double quotes. For example:

```
create table "1one"(col1 char(3))
create table "include spaces" (col1 int)

create table "grant"("add" int)
insert "grant"("add") values (3)
```

While the **quoted_identifier** option is turned on, do not use double quotes around character or date strings; use single quotes instead. Delimiting these strings with double quotes causes Adaptive Server to treat them as identifiers. For example, to insert a character string into *col1* of *1table*, use:

```
insert "1one"(col1) values ('abc')
```

not:

```
insert "1one"(col1) values ("abc")
```

To insert a single quote into a column, use two consecutive single quotation marks. For example, to insert the characters "a'b" into *col1* use:

```
insert "1one"(col1) values('a''b')
```

Using Qualified Object Names

You can uniquely identify a table or column by adding other names that qualify it—the database name, owner's name, and (for a column) the table or view name. Each qualifier is separated from the next one by a period. For example:

```
database.owner.table_name.column_name

database.owner.view_name.column_name
```

The naming conventions are:

```
[[database.]owner.]table_name
```

```
[[database.]owner.]view_name
```

Using Delimited Identifiers Within an Object Name

If you use **set quoted_identifier on**, you can use double quotes around individual parts of a qualified object name. Use a separate pair of quotes for each qualifier that requires quotes. For example, use:

```
database.owner."table_name"."column_name"
```

rather than:

```
database.owner."table_name.column_name"
```

Omitting the Owner Name

You can omit the intermediate elements in a name and use dots to indicate their positions, as long as the system is given enough information to identify the object:

```
database..table_name
database..view_name
```

Referencing Your Own Objects in the Current Database

You need not use the database name or owner name to reference your own objects in the current database. The default value for *owner* is the current user, and the default value for *database* is the current database.

If you reference an object without qualifying it with the database name and owner name, Adaptive Server tries to find the object in the current database among the objects you own.

Referencing Objects Owned by the Database Owner

If you omit the owner name and you do not own an object by that name, Adaptive Server looks for objects of that name owned by the Database Owner. You must qualify objects owned by the Database Owner only if you own an object of the same name, but you want to use the object owned by the Database Owner. However, you must qualify objects owned by other users with the user's name, whether or not you own objects of the same name.

Using Qualified Identifiers Consistently

When qualifying a column name and table name in the same statement, be sure to use the same qualifying expressions for each; they are evaluated as strings and must match; otherwise, an error is returned. The second of the following examples is incorrect because the syntax style for the column name does not match the syntax style used for the table name.

```
1. select demo.mary.publishers.city
   from demo.mary.publishers

   city
   ----------------------
   Boston
   Washington
   Berkeley

2. select demo.mary.publishers.city
   from demo..publishers

   The column prefix "demo.mary.publishers" does not
   match a table name or alias name used in the query.
```

Determining Whether an Identifier Is Valid

Use the system function **valid_name**, after changing character sets or before creating a table or view, to determine whether the object name is acceptable to Adaptive Server. Here is the syntax:

```
select valid_name("Object_name")
```

If *object_name* is not a valid identifier (for example, if it contains illegal characters or is more than 30 bytes long), Adaptive Server returns 0. If *object_name* is a valid identifier, Adaptive Server returns a nonzero number.

Renaming Database Objects

Rename user objects (including user-defined datatypes) with **sp_rename**.

◆ ***WARNING!***

After you rename a table or column, be sure to redefine any procedures, triggers, and views that depend on the renamed object.

Using Multibyte Character Sets

In multibyte character sets, a wider range of characters is available for use in identifiers. For example, on a server with the Japanese language installed, the following types of characters may be used as the first character of an identifier: Zenkaku or Hankaku Katakana, Hiragana, Kanji, Romaji, Greek, Cyrillic, or ASCII.

Although Hankaku Katakana characters are legal in identifiers on Japanese systems, they are not recommended for use in heterogeneous systems. These characters cannot be converted between the EUC-JIS and Shift-JIS character sets.

The same is true for some 8-bit European characters. For example, the character "Œ," the OE ligature, is part of the Macintosh character set (codepoint 0xCE). This character does not exist in the ISO 8859-1 (iso_1) character set. If "Œ" exists in data being converted from the Macintosh to the ISO 8859-1 character set, it causes a conversion error.

If an object identifier contains a character that cannot be converted, the client loses direct access to that object.

Pattern Matching with Wildcard Characters

Wildcard characters represent one or more characters, or a range of characters, in a *match_string*. A *match_string* is a character string containing the pattern to find in the expression. It can be any combination of constants, variables, and column names or a concatenated expression, such as:

```
like @variable + "%".
```

If the match string is a constant, it must always be enclosed in single or double quotes.

Use wildcard characters with the keyword **like** to find character and date strings that match a particular pattern. You cannot use **like** to search for seconds or milliseconds (see "Using Wildcard Characters with datetime Data" on page 3-22).

Use wildcard characters in **where** and **having** clauses to find character or date / time information that is **like**—or **not like**—the match string:

```
{where | having} [not]
   expression [not] like match_string
       [escape "escape_character"]
```

expression can be any combination of column names, constants, or functions with a character value.

Wildcard characters used without **like** have no special meaning. For example, this query finds any phone numbers that start with the four characters "415%":

```
select phone
from authors
where phone = "415%"
```

Using *not like*

Use **not like** to find strings that do not match a particular pattern. These two queries are equivalent: they find all the phone numbers in the *authors* table that do not begin with the 415 area code.

```
select phone
from authors
where phone not like "415%"

select phone
from authors
where not phone like "415%"
```

For example, this query finds the system tables in a database whose names begin with "sys":

```
select name
from sysobjects
where name like "sys%"
```

To see all the objects that are **not** system tables, use

```
not like "sys%"
```

If you have a total of 32 objects and **like** finds 13 names that match the pattern, **not like** will find the 19 objects that do not match the pattern.

not like and the negative wildcard character [^] may give different results (see "The Caret (^) Wildcard Character" on page 3-20). You cannot always duplicate **not like** patterns with **like** and **^**. This is because **not like** finds the items that do not match the entire **like** pattern, but **like** with negative wildcard characters is evaluated one character at a time.

A pattern such as **like "[^s][^y][^s]%"** may not produce the same results. Instead of 19, you might get only 14, with all the names that begin with "s" **or** have "y" as the second letter **or** have "s" as the third letter eliminated from the results, as well as the system table names. This is

because match strings with negative wildcard characters are evaluated in steps, one character at a time. If the match fails at any point in the evaluation, it is eliminated.

Case and Accent Insensitivity

If your Adaptive Server uses a case-insensitive sort order, case is ignored when comparing *expression* and *match_string*. For example, this clause:

```
where col_name like "Sm%"
```

would return "Smith," "smith," and "SMITH" on a case-insensitive Adaptive Server.

If your Adaptive Server is also accent-insensitive, it treats all accented characters as equal to each other and to their unaccented counterparts, both uppercase and lowercase. The **sp_helpsort** system procedure displays the characters that are treated as equivalent, displaying an "=" between them.

Using Wildcard Characters

You can use the match string with a number of wildcard characters, which are discussed in detail in the following sections. Table 3-7 summarizes the wildcard characters:

Table 3-7: Wildcard characters used with like

Symbol	Meaning
%	Any string of 0 or more characters
_	Any single character
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef])
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef])

Enclose the wildcard character and the match string in single or double quotes (**like** "[dD]eFr_nce").

The Percent Sign (%) Wildcard Character

Use the % wildcard character to represent any string of zero or more characters. For example, to find all the phone numbers in the *authors* table that begin with the 415 area code:

```
select phone
from authors
where phone like "415%"
```

To find names that have the characters “en” in them (Bennet, Green, McBadden):

```
select au_lname
from authors
where au_lname like "%en%"
```

Trailing blanks following “%” in a **like** clause are truncated to a single trailing blank. For example, “%” followed by two spaces matches “X ”(one space); “X ” (two spaces); “X   ” (three spaces), or any number of trailing spaces.

The Underscore (_) Wildcard Character

Use the _ wildcard character to represent any single character. For example, to find all six-letter names that end with “heryl” (for example, Cheryl):

```
select au_fname
from authors
where au_fname like "_heryl"
```

Bracketed ([ ]) Characters

Use brackets to enclose a range of characters, such as [a-f], or a set of characters such as [a2Br]. When ranges are used, all values in the sort order between (and including) *rangespec1* and *rangespec2* are returned. For example, “[0-z” matches 0-9, A-Z and a-z (and several punctuation characters) in 7-bit ASCII.

To find names ending with “inger” and beginning with any single character between M and Z:

```
select au_lname
from authors
where au_lname like "[M-Z]inger"
```

To find both “DeFrance” and “deFrance”:

```
select au_lname
from authors
where au_lname like "[dD]eFrance"
```

The Caret (^) Wildcard Character

The caret is the negative wildcard character. Use it to find strings that do not match a particular pattern. For example, "[^a-f]" finds strings that are not in the range a-f and "[^a2bR]" finds strings that are not "a," "2," "b," or "R."

To find names beginning with "M" where the second letter is not "c":

```
select au_lname
from authors
where au_lname like "M[^c]%"
```

When ranges are used, all values in the sort order between (and including) *rangespec1* and *rangespec2* are returned. For example, "[0-z]" matches 0-9, A-Z , a-z, and several punctuation characters in 7-bit ASCII.

Using Multibyte Wildcard Characters

If the multibyte character set configured on your Adaptive Server defines equivalent double-byte characters for the wildcard characters _, %, - [,], and ^, you can substitute the equivalent character in the match string. The underscore equivalent represents either a single- or double-byte character in the match string.

Using Wildcard Characters As Literal Characters

To search for the occurrence of %, _, [,], or ^ within a string, you must use an escape character. When a wildcard character is used in conjunction with an escape character, Adaptive Server interprets the wildcard character literally, rather than using it to represent other characters.

Adaptive Server provides two types of escape characters:

- Square brackets (a Transact-SQL extension)
- Any single character that immediately follows an **escape** clause (compliant with the SQL standards)

Using Square Brackets As Escape Characters

Use square brackets as escape characters for the percent sign, the underscore, and the left bracket. The right bracket does not need an escape character; use it by itself. If you use the hyphen as a literal character, it must be the first character inside a set of square brackets.

Table 3-8 shows some examples of square brackets as escape characters:

Table 3-8: Using square brackets to search for wildcard characters

like Predicate	Meaning
like "5%"	5 followed by any string of 0 or more characters
like "5[%]"	5%
like "_n"	an, in, on (and so on)
like "[_]n"	_n
like "[a-cdf]"	a, b, c, d, or f
like "[-acdf]"	-, a, c, d, or f
like "[[]"	[
like "]"	]
like "[[]ab]"	[]ab

Using the *escape* Clause

Use the **escape** clause to specify an escape character. Any single character in the server's default character set can be used as an escape character. If you try to use more than one character as an escape character, Adaptive Server generates an exception.

Do not use existing wildcard characters as escape characters because:

- If you specify the underscore (_) or percent sign (%) as an escape character, it loses its special meaning within that **like** predicate and acts only as an escape character.
- If you specify the left or right bracket ([or]) as an escape character, the Transact-SQL meaning of the bracket is disabled within that **like** predicate.
- If you specify the hyphen (-) or caret (^) as an escape character, it loses its special meaning and acts only as an escape character.

An escape character retains its special meaning within square brackets, unlike wildcard characters such as the underscore, the percent sign, and the open bracket.

The escape character is valid only within its **like** predicate and has no effect on other **like** predicates contained in the same statement. The only characters that are valid following an escape character are the wildcard characters (_, %, [,], or [^]), and the escape character itself. The escape character affects only the character following it, and subsequent characters are not affected by it.

If the pattern contains two literal occurrences of the character that happens to be the escape character, the string must contain four

consecutive escape characters. If the escape character does not divide the pattern into pieces of one or two characters, Adaptive Server returns an error message.

Following are examples of **like** predicates with **escape** clauses:.

Table 3-9: Using the escape clause

like Predicate	Meaning
like "5@%" escape "@"	5%
like "*_n" escape "*"	_n
like "%80@%%" escape "@"	String containing 80%
like "*_sql%" escape "*"**	String containing _sql*
like "%#####_#%%" escape "#"	String containing ##_%

Using Wildcard Characters with *datetime* Data

When you use **like** with *datetime* values, Adaptive Server converts the dates to the standard *datetime* format, then to *varchar*. Since the standard storage format does not include seconds or milliseconds, you cannot search for seconds or milliseconds with **like** and a pattern.

It is a good idea to use **like** when you search for *datetime* values, since *datetime* entries may contain a variety of date parts. For example, if you insert the value “9:20” and the current date into a column named *arrival_time*, the clause:

```
where arrival_time = '9:20'
```

would not find the value, because Adaptive Server converts the entry into “Jan 1 1900 9:20AM.” However, the following clause would find this value:

where arrival_time like '%9:20%'

4 Reserved Words

Keywords, also known as reserved words, are words that have special meanings. This chapter lists Transact-SQL and SQL92 keywords.

Transact-SQL Keywords

The words in the following list are reserved by Adaptive Server as keywords (part of SQL command syntax). They cannot be used as names of database objects such as databases, tables, rules, or defaults. They can be used as names of local variables and as stored procedure parameter names.

To find the names of existing objects that are reserved words, use **sp_checkreswords**.

A

add, all, alter, and, any, arith_overflow, as, asc, at, authorization, avg

B

begin, between, break, browse, bulk, by

C

cascade, case, char_convert, check, checkpoint, close, clustered, coalesce, commit, compute, confirm, connect, constraint, continue, controlrow, convert, count, create, current, cursor

D

database, dbcc, deallocate, declare, default, delete, desc, disk distinct, double, drop, dummy, dump

E

else, end, endtran, errlvl, errordata, errorexit, escape, except, exclusive, exec, execute, exists, exit, exp_row_size, external

F

fetch, fillfactor, for, foreign, from

G

goto, grant, group

H

having, holdlock

I

identity, identity_gap, identity_insert, identity_start, if, in, index, insert, install, intersect, into, is, isolation

J

jar, join

K

key, kill

L

level, like, lineno, load, lock

M

max, max_rows_per_page, min, mirror, mirrorexit, modify

N

national, noholdlock, nonclustered, not, null, nullif, numeric_truncation

O

of, off, offsets, on, once, online, only, open, option, or, order, over

P

partition, perm, permanent, plan, precision, prepare, primary, print

privileges, proc, procedure, processexit, proxy_table, public

Q

quiesce

R

raiserror, read, readpast, readtext, reconfigure, references remove, reorg, replace, replication, reservepagegap, return, revoke, role, rollback, rowcount, rows, rule

S

save, schema, select, set, setuser, shared, shutdown, some, statistics, stripe, sum, syb_identity, syb_restree

T

table, temp, temporary, textsize, to, tran, transaction, trigger, truncate, tsequal

U

union, unique, unpartition, update, use, user, user_option, using

V

values, varying, view

W

waitfor, when, where, while, with, work, writetext

SQL92 Keywords

Adaptive Server includes entry-level SQL92 features. Full SQL92 implementation includes the words listed in the following tables as command syntax. Upgrading identifiers can be a complex process; therefore, we are providing this list for your convenience. The publication of this information does not commit Sybase to providing all of these SQL92 features in subsequent releases. In addition, subsequent releases may include keywords not included in this list.

The words in the following list are SQL92 keywords that are not reserved words in Transact-SQL.

A

absolute, action, allocate, are, assertion

B

bit, bit_length, both

C

cascaded, case, cast, catalog, char, char_length, character, character_length, coalesce, collate, collation, column, connection, constraints, corresponding, cross, current_date, current_time, current_timestamp, current_user

D

date, day, dec, decimal, deferrable, deferred, describe, descriptor, diagnostics, disconnect, domain

E

end-exec, exception, extract

F

false, first, float, found, full

G

get, global, go

H

hour

I

immediate, indicator, initially, inner, input, insensitive, int, integer, interval

J

join

L

language, last, leading, left, local, lower

M

match, minute, module, month

N

names, natural, nchar, next, no, nullif, numeric

O

octet_length, outer, output, overlaps

P

pad, partial, position, preserve, prior

R

real, relative, restrict, right

S

scroll, second, section, session_user , size , smallint, space, sql, sqlcode, sqlerror, sqlstate, substring, system_user

T

then, time, timestamp, timezone_hour, timezone_minute, trailing, translate, translation, trim, true

U

unknown, upper, usage

V

value, varchar

W

when, whenever, write, year

Z

zone

Potential SQL92 Reserved Words

If you are using the ISO / IEC 9075:1989 standard, also avoid using the words shown in the following list because these words may become SQL92 reserved words in the future.

A

after, alias, async

B

before, boolean, breadth

C

call, completion, cycle

D

data, depth, dictionary

E

each, elseif, equals

G

general

I

ignore

L

leave, less, limit, loop

M

modify

N

new, none

O

object, oid, old, operation, operators, others

P

parameters, pendant, preorder, private, protected

R

recursive, ref, referencing, resignal, return, returns, routine, row

S

savepoint, search, sensitive, sequence, signal, similar, sqlexception, structure

T

test, there, type

U

under

V

variable, virtual, visible

W

wait, without

5 SQLSTATE Codes and Messages

This chapter describes Adaptive Server's SQLSTATE status codes and their associated messages. SQLSTATE codes are required for entry level SQL92 compliance. They provide diagnostic information about two types of conditions:

- **Warnings** – conditions that require user notification but are not serious enough to prevent a SQL statement from executing successfully
- **Exceptions** – conditions that prevent a SQL statement from having any effect on the database

Each SQLSTATE code consists of a 2-character class followed by a 3-character subclass. The class specifies general information about error type. The subclass specifies more specific information.

SQLSTATE codes are stored in the *sysmessages* system table, along with the messages that display when these conditions are detected. Not all Adaptive Server error conditions are associated with a SQLSTATE code—only those mandated by SQL92. In some cases, multiple Adaptive Server error conditions are associated with a single SQLSTATE value.

Warnings

Adaptive Server currently detects only one SQLSTATE warning condition, which is described in Table 5-1:

Table 5-1: SQLSTATE warnings

Message	Value	Description
Warning - null value eliminated in set function.	01003	Occurs when you use an aggregate function (**avg**, **max**, **min**, **sum**, or **count**) on an expression with a null value.

Exceptions

Adaptive Server detects the following types of exceptions:

- Cardinality violations
- Data exceptions

- Integrity constraint violations
- Invalid cursor states
- Syntax errors and access rule violations
- Transaction rollbacks
- **with check option** violations

Exception conditions are described in Table 5-2 through Table 5-8. Each class of exceptions appears in its own table. Within each table, conditions are sorted alphabetically by message text.

Cardinality Violations

Cardinality violations occur when a query that should return only a single row returns more than one row to an Embedded SQL™ application.

Table 5-2: Cardinality violations

Message	Value	Description
Subquery returned more than 1 value. This is illegal when the subquery follows =, !=, <, <=, >, >=. or when the subquery is used as an expression.	21000	Occurs when: • A scalar subquery or a row subquery returns more than one row. • A **select into parameter_list** query in Embedded SQL returns more than one row.

Data Exceptions

Data exceptions occur when an entry:

- Is too long for its datatype,
- Contains an illegal escape sequence, or
- Contains other format errors.

Table 5-3: Data exceptions

Message	Value	Description
Arithmetic overflow occurred.	22003	Occurs when: • An exact numeric type would lose precision or scale as a result of an arithmetic operation or **sum** function. • An approximate numeric type would lose precision or scale as a result of truncation, rounding, or a **sum** function.
Data exception - string data right truncated.	22001	Occurs when a *char* or *varchar* column is too short for the data being inserted or updated and non-blank characters must be truncated.
Divide by zero occurred.	22012	Occurs when a numeric expression is being evaluated and the value of the divisor is zero.
Illegal escape character found. There are fewer bytes than necessary to form a valid character.	22019	Occurs when you are searching for strings that match a given pattern if the escape sequence does not consist of a single character.
Invalid pattern string. The character following the escape character must be percent sign, underscore, left square bracket, right square bracket, or the escape character.	22025	Occurs when you are searching for strings that match a particular pattern when: • The escape character is not immediately followed by a percent sign, an underscore, or the escape character itself, or • The escape character partitions the pattern into substrings whose lengths are other than 1 or 2 characters.

Integrity Constraint Violations

Integrity constraint violations occur when an **insert**, **update**, or **delete** statement violates a **primary key**, **foreign key**, **check**, or **unique** constraint or a unique index.

Table 5-4: Integrity constraint violations

Message	Value	Description
Attempt to insert duplicate key row in object *object_name* with unique index *index_name*	23000	Occurs when a duplicate row is inserted into a table that has a unique constraint or index.

Table 5-4: Integrity constraint violations (continued)

Message	Value	Description
Check constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **update** or **delete** would violate a check constraint on a column.
Dependent foreign key constraint violation in a referential integrity constraint. dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **update** or **delete** on a primary key table would violate a foreign key constraint.
Foreign key constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **insert** or **update** on a foreign key table is performed without a matching value in the primary key table.

Invalid Cursor States

Invalid cursor states occur when:

- A **fetch** uses a cursor that is not currently open, or
- An **update where current of** or **delete where current of** affects a cursor row that has been modified or deleted, or
- An **update where current of** or **delete where current of** affects a cursor row that not been fetched.

Table 5-5: Invalid cursor states

Message	Value	Description
Attempt to use cursor *cursor_name* which is not open. Use the system stored procedure sp_cursorinfo for more information.	24000	Occurs when an attempt is made to fetch from a cursor that has never been opened or that was closed by a **commit** statement or an implicit or explicit **rollback**. Reopen the cursor and repeat the **fetch**.
Cursor *cursor_name* was closed implicitly because the current cursor position was deleted due to an update or a delete. The cursor scan position could not be recovered. This happens for cursors which reference more than one table.	24000	Occurs when the join column of a multitable cursor has been deleted or changed. Issue another **fetch** to reposition the cursor.

Table 5-5: Invalid cursor states (continued)

Message	Value	Description
The cursor *cursor_name* had its current scan position deleted because of a DELETE / UPDATE WHERE CURRENT OF or a regular searched DELETE / UPDATE. You must do a new FETCH before doing an UPDATE or DELETE WHERE CURRENT OF.	24000	Occurs when a user issues an **update/delete where current of** whose current cursor position has been deleted or changed. Issue another **fetch** before retrying the **update/delete where current of**.
The UPDATE / DELETE WHERE CURRENT OF failed for the cursor *cursor_name* because it is not positioned on a row.	24000	Occurs when a user issues an **update/delete where current of** on a cursor that: • Has not yet fetched a row • Has fetched one or more rows after reaching the end of the result set

Syntax Errors and Access Rule Violations

Syntax errors are generated by SQL statements that contain unterminated comments, implicit datatype conversions not supported by Adaptive Server or other incorrect syntax.

Access rule violations are generated when a user tries to access an object that does not exist or one for which he or she does not have the correct permissions.

Table 5-6: Syntax errors and access rule violations

Message	Value	Description
command permission denied on object *object_name*, database *database_name*, owner *owner_name*.	42000	Occurs when a user tries to access an object for which he or she does not have the proper permissions.
Implicit conversion from datatype '*datatype*' to '*datatype*' is not allowed. Use the CONVERT function to run this query.	42000	Occurs when the user attempts to convert one datatype to another but Adaptive Server cannot do the conversion implicitly.
Incorrect syntax near *object_name*.	42000	Occurs when incorrect SQL syntax is found near the object specified.
Insert error: column name or number of supplied values does not match table definition.	42000	Occurs during inserts when an invalid column name is used or when an incorrect number of values is inserted.
Missing end comment mark '*/'.	42000	Occurs when a comment that begins with the /* opening delimiter does not also have the */ closing delimiter.

Table 5-6: Syntax errors and access rule violations (continued)

Message	Value	Description
object_name not found. Specify owner.objectname or use sp_help to check whether the object exists (sp_help may produce lots of output).	42000	Occurs when a user tries to reference an object that he or she does not own. When referencing an object owned by another user, be sure to qualify the object name with the name of its owner.
The size (*size*) given to the *object_name* exceeds the maximum. The largest size allowed is *size*.	42000	Occurs when: • The total size of all the columns in a table definition exceeds the maximum allowed row size. • The size of a single column or parameter exceeds the maximum allowed for its datatype.

Transaction Rollbacks

Transaction rollbacks occur when the **transaction isolation level** is set to 3, but Adaptive Server cannot guarantee that concurrent transactions can be serialized. This type of exception generally results from system problems such as disk crashes and offline disks.

Table 5-7: Transaction rollbacks

Message	Value	Description
Your server command (process id #*process_id*) was deadlocked with another process and has been chosen as deadlock victim. Re-run your command.	40001	Occurs when Adaptive Server detects that it cannot guarantee that two or more concurrent transactions can be serialized.

with check option Violation

This class of exception occurs when data being inserted or updated through a view would not be visible through the view.

Table 5-8: with check option violation

Message	Value	Description
The attempted insert or update failed because the target view was either created WITH CHECK OPTION or spans another view created WITH CHECK OPTION. At least one resultant row from the command would not qualify under the CHECK OPTION constraint.	44000	Occurs when a view, or any view on which it depends, was created with a **with check option** clause.

For the Index, see volume 4, “Tables and Reference Manual Index.”

Volume 4, “Tables and Reference Manual Index,” contains the index entries for all volumes of the *Adaptive Server Reference Manual*.

For the Index, see volume 4, “Tables and Reference Manual Index.”

Sybase® Adaptive Server™ Enterprise Reference Manual

Volume 2: Commands

Adaptive Server Enterprise Version 12
Document ID: 36272-01-1200-01
Last Revised: October 1999

Principal author: Enterprise Data Studios Publications

Contributing authors: Anneli Meyer, Evelyn Wheeler

Document ID: 36272-01-1200

This publication pertains to Adaptive Server Enterprise Version 12 of the Sybase database management software and to any subsequent version until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Sybase, the SYBASE logo, Adaptive Server, APT-FORMS, Certified SYBASE Professional, the Certified SYBASE Professional logo, Column Design, ComponentPack, Data Workbench, First Impression, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, Transact-SQL, Visual Components, VisualWriter, and VQL are registered trademarks of Sybase, Inc.

Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server Enterprise Monitor, Adaptive Warehouse, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, EnterpriseConnect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript,

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

About This Book

6. Transact-SQL Commands

For the Index, see volume 4, "Tables and Reference Manual Index."

List of Figures

List of Tables

About This Book

The *Adaptive Server Reference Manual* is a four-volume guide to Sybase® Adaptive Server™ Enterprise and the Transact-SQL® language.

Volume 1, "*Building Blocks*," describes the "parts" of Transact-SQL: datatypes, built-in functions, expressions and identifiers, SQLSTATE errors, and reserved words. Before you can use Transact-SQL sucessfully, you need to understand the purpose of each of these building blocks and how its use affects the results of Transact-SQL statements.

Volume 2, "*Commands*," provides reference information about the Transact-SQL commands, which you use to create statements.

Volume 3, "*Procedures*" provides reference information about system procedures, catalog stored procedures, extended stored procedures, and **dbcc** stored procedures. All procedures are created using Transact-SQL statements.

Volume 4, "*Tables and Reference Manual Index*," provides reference information about the system tables, which store information about your server, databases, users, and other information. It also provides information about the tables in the *dbccdb* and *dbccalt* databases. It also contains an index that covers the topics of all four volumes.

For information about the intended audience of this book, related documents, other sources of information, conventions used in this manual, and help, see "About This Book" in Volume 1.

How to Use This Book

This manual contains:

- Chapter 6, "Transact-SQL Commands," which provides reference information for every Transact-SQL command. Particularly complex commands, such as **select**, are divided into subsections. For example, there are reference pages on the **compute** clause and on the **group by** and **having** clauses of the **select** command.

6 Transact-SQL Commands

This chapter describes commands, clauses, and other elements used to construct a Transact-SQL statement.

Overview

Table 6-1 provides a brief description of the commands in this chapter.

Table 6-1: Transact-SQL commands

Command	Description
alter database	Increases the amount of space allocated to a database.
alter role	Defines mutually exclusive relationships between roles and adds, drops, and changes passwords for roles.
alter table	Adds new columns; adds, changes, or drops constraints, changes constraints; partitions or unpartitions an existing table.
begin...end	Encloses a series of SQL statements so that control-of-flow language, such as **if...else**, can affect the performance of the whole group.
begin transaction	Marks the starting point of a user-defined transaction.
break	Causes an exit from a **while** loop. **break** is often activated by an **if** test.
case	Allows SQL expressions to be written for conditional values. **case** expressions can be used anywhere a value expression can be used.
checkpoint	Writes all **dirty** pages (pages that have been updated since they were last written) to the database device.
close	Deactivates a cursor.
coalesce	Allows SQL expressions to be written for conditional values. **coalesce** expressions can be used anywhere a value expression can be used; alternative for a **case** expression.
commit	Marks the ending point of a user-defined transaction.
compute Clause	Generates summary values that appear as additional rows in the query results.

Table 6-1: Transact-SQL commands (continued)

Command	Description
connect to...disconnect	Specifies the server to which a passthrough connection is required.
continue	Causes the **while** loop to restart. **continue** is often activated by an **if** test.
create database	Creates a new database.
create default	Specifies a value to insert in a column (or in all columns of a user-defined datatype) if no value is explicitly supplied at insert time.
create existing table	Confirms that the current remote table information matches the information that is stored in *column_list*, and verifies the existence of the underlying object.
create index	Creates an index on one or more columns in a table.
create plan	Creates an abstract query plan.
create procedure	Creates a stored procedure that can take one or more user-supplied parameters.
create proxy_table	Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.
create role	Creates a user-defined role.
create rule	Specifies the domain of acceptable values for a particular column or for any column of a user-defined datatype.
create schema	Creates a new collection of tables, views and permissions for a database user.
create table	Creates new tables and optional integrity constraints.
create trigger	Creates a trigger, a type of stored procedure often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.
create view	Creates a view, which is an alternative way of looking at the data in one or more tables.
dbcc	Database Consistency Checker (**dbcc**) checks the logical and physical consistency of a database. Use **dbcc** regularly as a periodic check or if you suspect any damage.
deallocate cursor	Makes a cursor inaccessible and releases all memory resources committed to that cursor.

Table 6-1: Transact-SQL commands (continued)

Command	Description
declare	Declares the name and type of local variables for a batch or procedure.
declare cursor	Defines a cursor.
delete	Removes rows from a table.
delete statistics	Removes statistics from the *sysstatistics* system table.
disk init	Makes a physical device or file usable by Adaptive Server.
disk mirror	Creates a software mirror that immediately takes over when the primary device fails.
disk refit	Rebuilds the *master* database's *sysusages* and *sysdatabases* system tables from information contained in *sysdevices*. Use **disk refit** after **disk reinit** as part of the procedure to restore the *master* database.
disk reinit	Rebuilds the *master* database's *sysdevices* system table. Use **disk reinit** as part of the procedure to restore the *master* database.
disk remirror	Reenables disk mirroring after it is stopped by failure of a mirrored device or temporarily disabled by the **disk unmirror** command.
disk unmirror	Disables either the original device or its mirror, allowing hardware maintenance or the changing of a hardware device.
drop database	Removes one or more databases from a Adaptive Server.
drop default	Removes a user-defined default.
drop index	Removes an index from a table in the current database.
drop procedure	Removes user-defined stored procedures.
drop role	Removes a user-defined role.
drop rule	Removes a user-defined rule.
drop table	Removes a table definition and all of its data, indexes, triggers, and permission specifications from the database.
drop trigger	Removes a trigger.
drop view	Removes one or more views from the current database.

Table 6-1: Transact-SQL commands (continued)

Command	Description
dump database	Makes a backup copy of the entire database, including the transaction log, in a form that can be read in with **load database**. Dumps and loads are performed through Backup Server.
dump transaction	Makes a copy of a transaction log and removes the inactive portion.
execute	Runs a system procedure, a user-defined stored procedure, or a dynamically constructed Transact-SQL command.
fetch	Returns a row or a set of rows from a cursor result set.
goto Label	Branches to a user-defined label.
grant	Assigns permissions to users or to user-defined roles.
group by and having Clauses	Used in **select** statements to divide a table into groups and to return only groups that match conditions in the having clause.
if...else	Imposes conditions on the execution of a SQL statement.
insert	Adds new rows to a table or view.
kill	Kills a process.
load database	Loads a backup copy of a user database, including its transaction log.
load transaction	Loads a backup copy of the transaction log.
lock table	Explicitly locks a table within a transaction.
nullif	Allows SQL expressions to be written for conditional values. **nullif** expressions can be used anywhere a value expression can be used; alternative for a **case** expression.
online database	Marks a database available for public use after a normal load sequence and, if needed, upgrades a loaded database and transaction log dumps to the current version of Adaptive Server.
open	Opens a cursor for processing.
order by Clause	Returns query results in the specified column(s) in sorted order.
prepare transaction	Used by DB-Library™ in a two-phase commit application to see if a server is prepared to commit a transaction.
print	Prints a user-defined message on the user's screen.

Table 6-1: Transact-SQL commands (continued)

Command	Description
quiesce database	Suspends and resumes updates to a specified list of databases.
raiserror	Prints a user-defined error message on the user's screen and sets a system flag to record that an error condition has occurred.
readtext	Reads *text* and *image* values, starting from a specified offset and reading a specified number of bytes or characters.
reconfigure	The **reconfigure** command currently has no effect; it is included to allow existing scripts to run without modification. In previous releases, **reconfigure** was required after the **sp_configure** system procedure to implement new configuration parameter settings.
remove java	Removes one or more Java-SQL classes, packages, or JARs from a database. Use when Java is enabled in the database.
reorg	Reclaims unused space on pages, removes row forwarding, or rewrites all rows in the table to new pages, depending on the option used.
return	Exits from a batch or procedure unconditionally, optionally providing a return status. Statements following **return** are not executed.
revoke	Revokes permissions or roles from users or roles.
rollback	Rolls a user-defined transaction back to the last savepoint inside the transaction or to the beginning of the transaction.
rollback trigger	Rolls back the work done in a trigger, including the update that caused the trigger to fire, and issues an optional **raiserror** statement.
save transaction	Sets a savepoint within a transaction.
select	Retrieves rows from database objects.
set	Sets Adaptive Server query-processing options for the duration of the user's work session. Can be used to set some options inside a trigger or stored procedure. Can also be used to activate or deactivate a role in the current session.
setuser	Allows a Database Owner to impersonate another user.

Table 6-1: Transact-SQL commands (continued)

Command	Description
shutdown	Shuts down Adaptive Server or a Backup Server™. This command can be issued only by a System Administrator.
truncate table	Removes all rows from a table.
union Operator	Returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the **all** keyword is specified.
update	Changes data in existing rows, either by adding data or by modifying existing data; updates all statistics information for a given table; updates information about the number of pages in each partition for a partitioned table; updates information about the distribution of key values in specified indexes.
use	Specifies the database with which you want to work.
waitfor	Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.
where Clause	Sets the search conditions in a **select**, **insert**, **update**, or **delete** statement.
while	Sets a condition for the repeated execution of a statement or statement block. The statement(s) execute repeatedly, as long as the specified condition is true.
writetext	Permits non-logged, interactive updating of an existing *text* or *image* column.

alter database

Function

Increases the amount of space allocated to a database.

Syntax

```
alter database database_name
   [on {default | database_device } [= size]
       [, database_device [= size]]...]
   [log on { default | database_device } [ = size ]
       [ , database_device [= size]]...]
   [with override]
   [for load]
   [for proxy_update]]
```

Keywords and Options

database_name – is the name of the database. The database name can be a literal, a variable, or a stored procedure parameter.

on – indicates a size and / or location for the database extension. If you have your log and data on separate device fragments, use this clause for the data device and the **log on** clause for the log device.

default – indicates that **alter database** can put the database extension on any default database device(s) (as shown by **sp_helpdevice**). To specify a size for the database extension without specifying the exact location, use this command:

```
on default = size
```

To change a database device's status to default, use the system procedure **sp_diskdefault**.

database_device – is the name of the database device on which to locate the database extension. A database can occupy more than one database device with different amounts of space on each. Add database devices to Adaptive Server with **disk init**.

size – is the amount of space, in megabytes, to allocate to the database extension. The minimum extension is 1MB (512 2K pages). The default value is 2MB.

log on – indicates that you want to specify additional space for the database's transaction logs. The **log on** clause uses the same defaults as the **on** clause.

with override – forces Adaptive Server to accept your device specifications, even if they mix data and transaction logs on the same device, thereby endangering up-to-the-minute recoverability for your database. If you attempt to mix log and data on the same device without using this clause, the **alter database** command fails. If you mix log and data, and use **with override**, you are warned, but the command succeeds.

for load – is used only after **create database for load**, when you must re-create the space allocations and segment usage of the database being loaded from a dump.

for proxy_update – forces the re-synchronization of proxy tables within the proxy database.

Examples

1.
```
alter database mydb
```

   Adds 1MB to the database *mydb* on a default database device.

2.
```
alter database pubs2
on newdata = 3
```

   Adds 3MB to the space allocated for the *pubs2* database on the database device named *newdata*.

3.
```
alter database production
on userdata1 = 10
log on logdev = 2
```

   Adds 10MB of space for data on *userdata1* and 2MB for the log on *logdev*.

Comments

Restrictions

- You must be using the *master* database, or executing a stored procedure in the *master* database, to use **alter database**.
- If Adaptive Server cannot allocate the requested space, it comes as close as possible per device and prints a message telling how much space has been allocated on each database device.
- You can expand the *master* database only on the master device. An attempt to use **alter database** to expand the *master* database to

any other database device results in an error message. Here is an example of the correct statement for modifying the *master* database on the master device:

```
alter database master on master = 1
```

- The maximum number of device fragments for any database is 128. Each time you allocate space on a database device with **create database** or **alter database**, that allocation represents a device fragment, and the allocation is entered as a row in *sysusages*.
- If you use **alter database** on a database that is in the process of being dumped, the **alter database** command cannot complete until the dump finishes. Adaptive Server locks the in-memory map of database space use during a dump. If you issue an **alter database** command while this in-memory map is locked, Adaptive Server updates the map from the disk after the dump completes. If you interrupt **alter database**, Adaptive Server instructs you to run **sp_dbremap**. If you fail to run **sp_dbremap**, the space you added does not become available to Adaptive Server until the next reboot.
- You can use **alter database on** *database_device* on an offline database.

Backing Up *master* After Allocating More Space

- Back up the *master* database with the **dump database** command after each use of **alter database**. This makes recovery easier and safer in case *master* becomes damaged.
- If you use **alter database** and fail to back up *master*, you may be able to recover the changes with **disk refit**.

Placing the Log on a Separate Device

- To increase the amount of storage space allocated for the transaction log when you have used the **log on** extension to **create database**, give the name of the log's device in the **log on** clause when you issue the **alter database** command.
- If you did not use the **log on** extension of **create database** to place your logs on a separate device, you may not be able to recover fully in case of a hard disk crash. In this case, you can extend your logs by using **alter database** with the **log on** clause, then using **sp_logdevice**.

Getting Help on Space Usage

- To see the names, sizes, and usage of device fragments already in use by a database, execute **sp_helpdb** *dbname*.

- To see how much space the current database is using, execute **sp_spaceused**.

The *system* and *default* Segments

- The *system* and *default* segments are mapped to each new database device included in the **on** clause of an **alter database** command. To unmap these segments, use **sp_dropsegment**.
- When you use **alter database** (without **override)** to extend a database on a device already in use by that database, the segments mapped to that device are also extended. If you use the **override** clause, all device fragments named in the **on** clause become system / default segments, and all device fragments named in the **log on** clause become log segments.

Using *alter database* to Awaken Sleeping Processes

- If user processes are suspended because they have reached a last-chance threshold on a log segment, use **alter database** to add space to the log segment. The processes awaken when the amount of free space exceeds the last-chance threshold.

Using *for proxy_update*

- If the **for proxy_update** clause is entered with no other options, the size of the database will not be extended; instead, the proxy tables, if any, will be dropped from the proxy database and re-created from the metadata obtained from the pathname specified during **create database ... with default_location = 'pathname'.**
- If this command is used with other options to extend the size of the database, the proxy table synchronization is performed after the size extensions are made.
- The purpose of this **alter database** extension is to provide the DBA with an easy-to-use, single-step operation with which to obtain an accurate and up-to-date proxy representation of all tables at a single remote site.
- This re-synchronization is supported for all external data sources, and not just the primary server in a HA-cluster environment. Also, a database need not have been created with the **for proxy_update** clause. If a default storage location has been specified, either through the **create database** command or with **sp_defaultloc**, the metadata contained within the database can be synchronized with the metadata at the remote storage location.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

alter database permission defaults to the Database Owner. System Administrators can also alter databases.

See Also

Commands	**create database**, **disk init**, **drop database**, **load database**
System procedures	**sp_addsegment**, **sp_dropsegment**, **sp_helpdb**, **sp_helpsegment**, **sp_logdevice**, **sp_renamedb**, **sp_spaceused**

alter role

Function

Defines mutually exclusive relationships between roles; adds, drops, and changes passwords for roles; specifies the password expiration interval, the minimum password length, and the maximum number of failed logins allowed for a specified role

Syntax

```
alter role role1 { add | drop } exclusive {
   membership | activation } role2

alter role role_name [add passwd "password" |
   drop passwd] [lock | unlock]

alter role { role_name | "all overrides" }
   set { passwd expiration | min passwd length |
   max failed_logins } option_value
```

Keywords and Options

role1 – is one role in a mutually exclusive relationship.

add – adds a role in a mutually exclusive relationship; adds a password to a role.

drop – drops a role in a mutually exclusive relationship; drops a password from a role.

exclusive – makes both named roles mutually exclusive.

membership – does not allow you to grant users both roles at the same time.

activation – allows you to grant a user both roles at the same time, but does not allow the user to activate both roles at the same time.

role2 – is the other role in a mutually exclusive relationship.

role_name – is the name of the role for which you want to add, drop, or change a password.

passwd – adds a password to a role.

password – is the password to add to a role. Passwords must be at least 6 characters in length and must conform to the rules for identifiers. You cannot use variables for passwords.

lock locks the specified role.

unlock unlocks the specified role.

all overrides applies the setting that follows to the entire server rather than to a specific role.

set activates the option that follows it.

passwd expiration specifies the password expiration interval in days. It can be any value between 0 and 32767, inclusive.

min passwd length specifies the minimum length allowed for the specified password.

max failed_logins specifies the maximum number of failed login attempts allowed for the specified password.

option_value specifies the value for **passwd expiration**, **min passwd length**, or **max failed_logins**. To set **all overrides**, set the value of *option_value* to -1.

Examples

1. **`alter role intern_role add exclusive membership specialist_role`**

   Defines *intern_role* and *specialist_role* as mutually exclusive.

2. **`alter role specialist_role add exclusive membership intern_role`**
   `alter role intern_role add exclusive activation surgeon_role`

   Defines roles as mutually exclusive at the membership level and at the activation level.

3. **`alter role doctor_role add passwd "physician"`**

   Adds a password to an existing role.

4. **`alter role doctor_role drop passwd`**

   Drops a password from an existing role.

5. **`alter role physician_role lock`**

   Locks the role *physician_role*.

6. **`alter role physician_role unlock`**

   Unlocks the role *physician_role*.

7. **`alter role physician_role set max failed_logins 5`**

   Changes the maximum number of failed logins allowed for *physician_role* to 5.

8. **`alter role physician_role set min passwd length 5`**

Sets the minimum password length for *physician_role*, an existing role, to five characters.

```
9. alter role "all overrides" set min passwd length -1
```

Overrides the minimum password length of all roles.

```
10.alter role "all overrides" set max failed_logins -1
```

Removes the overrides for the maximum failed logins for all roles.

Comments

- The **alter role** command defines mutually exclusive relationships between roles and adds, drops, and changes passwords for roles.
- For more information on altering roles, see the *System Administration Guide.*
- The **all overrides** parameter removes the system overrides that were set using **sp_configure** with any of the following parameters:
  - **passwd expiration**
  - **max failed_logins**
  - **min passwd length**

  Dropping the role password removes the overrides for the password expiration and the maximum failed logins options.

Mutually Exclusive Roles

- You need not specify the roles in a mutually exclusive relationship or role hierarchy in any particular order.
- You can use mutual exclusivity with role hierarchy to impose constraints on user-defined roles.
- Mutually exclusive membership is a stronger restriction than mutually exclusive activation. If you define two roles as mutually exclusive at membership, they are implicitly mutually exclusive at activation.
- If you define two roles as mutually exclusive at membership, defining them as mutually exclusive at activation has no effect on the membership definitions. Mutual exclusivity at activation is added and dropped independently of mutual exclusivity at membership.
- You cannot define two roles as having mutually exclusive after granting both roles to users or roles. Revoke either granted role

from existing grantees before attempting to define the roles as mutually exclusive on the membership level.

- If two roles are defined as mutually exclusive at activation, the System Security Officer can assign both roles to the same user, but the user cannot activate both roles at the same time.
- If the System Security Officer defines two roles as mutually exclusive at activation, and users have already activated both roles or, by default, have set both roles to activate at login, Adaptive Server makes the roles mutually exclusive, but issues a warning message naming specific users with conflicting roles. The users' activated roles do not change.

Changing Passwords for Roles

- To change the password for a role, first drop the existing password, then add the new password, as follows:

```
alter role doctor_role drop passwd
alter role doctor_role add passwd "physician"
```

➤ ***Note***

Passwords attached to user-defined roles do not expire.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Security Officer can execute **alter role**.

See Also

Commands	**create role**, **drop role**, **grant**, **revoke**, **set**
Functions	**mut_excl_roles**, **proc_role**, **role_contain**, **role_id**, **role_name**
System Procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_modifylogin**

alter table

Function

Adds new columns; drops, modifies existing columns; adds, changes, or drops constraints; partitions or unpartitions an existing table; changes the locking scheme for an existing table; specifies ascending or descending index order when **alter table** is used to create referential integrity constraints that are based on indexes; specifies the ratio of filled pages to empty pages, to reduce storage fragmentation.

Syntax

```
alter table [database.[owner].]table_name

{add column_name datatype
    [default {constant_expression | user | null}]
    {identity | null | not null}
    [off row | in row]
    [ [constraint constraint_name]
      { { unique | primary key }
          [clustered | nonclustered] [asc | desc]
          [with { { fillfactor = pct
                  | max_rows_per_page = num_rows }
                  , reservepagegap = num_pages }]
          [on segment_name]
        | references [[database.]owner.]ref_table
          [(ref_column)]
        | check (search_condition) ] ... }
     [, next_column]...

| add { [constraint constraint_name]
      { {unique | primary key}
          [clustered | nonclustered]
          (column_name [asc | desc]
                [, column_name [asc | desc]...])
        [with { { fillfactor = pct
                | max_rows_per_page = num_rows}
                , reservepagegap = num_pages}]
        [on segment_name]
   | foreign key (column_name [{, column_name}...])
        references [[database.]owner.]ref_table
            [(ref_column [{, ref_column}...])]
    | check (search_condition)}

| drop {[column_name [, column_name]] |
    [constraint constraint_name]}
```

```
| modify column_name {[datatype] [null] |
    [not null]] [, column_name]

| replace column_name
    default {constant_expression | user | null}

| partition number_of_partitions

| unpartition

| enable | disable trigger

| lock {allpages | datarows | datapages } }

| with exp_row_size = num_bytes
```

Keywords and Options

table_name – is the name of the table to change. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

add – specifies the name of the column or constraint to add to the table.

If Component Integration Services is enabled, you cannot use **add** for remote servers.

column_name – is the name of a column in that table. If Java is enabled in the database, the column can be a Java-SQL column.

datatype – is any system datatype except *bit* or any user-defined datatype except those based on *bit*.

If Java is enabled in the database, can be the name of a Java class installed in the database, either a system class or a user-defined class. Refer to *Java in Adaptive Server Enterprise* for more information.

default – specifies a default value for a column. If you specify a default and the user does not provide a value for this column when inserting data, Adaptive Server inserts this value. The default can be a *constant_expression*, **user** (to insert the name of the user who is inserting the data), or **null** (to insert the null value).

Adaptive Server generates a name for the default in the form of *tabname_colname_objid*, where *tabname* is the first 10 characters of the table name, *colname* is the first 5 characters of the column name, and *objid* is the object ID number for the default. Setting the default to **null** drops the default.

If Component Integration Services is enabled, you cannot use **default** for remote servers.

constant_expression – is a constant expression to use as a default value for a column. It cannot include the name of any columns or other database objects, but can include built-in functions. This default value must be compatible with the datatype of the column.

user – specifies that Adaptive Server should insert the user name as the default if the user does not supply a value. The datatype of the column must be either *char*(30), *varchar*(30), or a type that Adaptive Server implicitly converts to *char*; however, if the datatype is not *char*(30) or *varchar*(30), truncation may occur.

null | **not null** – specifies Adaptive Server's behavior during data insertion if no default exists.

null specifies that a column is added that allows nulls. Adaptive Server assigns a null value during inserts if a user does not provide a value.

not null specifies that a column is added that does not allow nulls. Users must provide a non-null value during inserts if no default exists.

If you do not specify **null** or **not null**, Adaptive Server uses **not null** by default. However, you can switch this default using **sp_dboption** to make the default compatible with the SQL standards. If you specify (or imply) **not null** for the newly added column, a default clause is required. The default value is used for all existing rows of the newly added column, and applies to future inserts as well.

identity – indicates that the column has the IDENTITY property. Each table in a database can have one IDENTITY column of type *numeric* and scale zero. IDENTITY columns are not updatable and do not allow nulls.

IDENTITY columns store sequential numbers, such as invoice numbers or employee numbers, automatically generated by Adaptive Server. The value of the IDENTITY column uniquely identifies each row in a table.

If Component Integration Services is enabled, you cannot use **identity** for remote servers.

off row | **in row** – specifies whether the Java-SQL column is stored separate from the row or in storage allocated directly in the row.

The storage for an **in row** column must not exceed 255 bytes. The default value is **off row.**

constraint – introduces the name of an integrity constraint.

If Component Integration Services is enabled, you cannot use **constraint** for remote servers.

constraint_name – is the name of the constraint. It must conform to the rules for identifiers and be unique in the database. If you do not specify the name for a table-level constraint, Adaptive Server generates a name in the form of *tabname_colname_objectid*, where *tabname* is the first 10 characters of the table name, *colname* is the first 5 characters of the column name, and *objectid* is the object ID number for the constraint. If you do not specify the name for a unique or primary key constraint, Adaptive Server generates a name in the format *tabname_colname_tabindid*, where *tabindid* is a string concatenation of the table ID and index ID.

Constraints do not apply to the data that already exists in the table at the time the constraint is added.

unique – constrains the values in the indicated column or columns so that no two rows can have the same non-null value. This constraint creates a unique index that can be dropped only if the constraint is dropped. You cannot use this option along with the **null** option described above.

primary key – constrains the values in the indicated column or columns so that no two rows can have the same value and so that the value cannot be NULL. This constraint creates a unique index that can be dropped only if the constraint is dropped.

clustered | **nonclustered** – specifies that the index created by a **unique** or **primary key** constraint is a clustered or nonclustered index. **clustered** is the default (unless a clustered index already exists for the table) for primary key constraints; **nonclustered** is the default for unique constraints. There can be only one clustered index per table. See **create index** for more information.

fillfactor – specifies how full to make each page when Adaptive Server creates a new index on existing data. The **fillfactor** percentage is

relevant only when the index is created. As the data changes, the pages are not maintained at any particular level of fullness.

The default for **fillfactor** is 0; this is used when you do not include **with fillfactor** in the **create index** statement (unless the value has been changed with **sp_configure**). When specifying a **fillfactor**, use a value between 1 and 100.

A **fillfactor** of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes. There is seldom a reason to change the **fillfactor**.

If the **fillfactor** is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A **fillfactor** of 100 makes sense only for read-only tables—tables to which no additional data will ever be added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A **fillfactor** of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small **fillfactor** values cause each index (or index and data) to take more storage space.

◆ ***WARNING!***

Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

max_rows_per_page – limits the number of rows on data pages and the leaf level pages of indexes. Unlike **fillfactor**, the **max_rows_per_page** value is maintained until it is changed with **sp_chgattribute**.

If you do not specify a value for **max_rows_per_page**, Adaptive Server uses a value of 0 when creating the index. When specifying **max_rows_per_page** for data pages, use a value between 0 and 256. The maximum number of rows per page for nonclustered indexes depends on the size of the index key; Adaptive Server returns an error message if the specified value is too high.

For indexes created by constraints, a **max_rows_per_page** setting of 0 creates clustered indexes with full pages and nonclustered indexes with full leaf pages. A setting of 0 leaves a comfortable

amount of space within the index B-tree in both clustered and nonclustered indexes.

If **max_rows_per_page** is set to 1, Adaptive Server creates both clustered and nonclustered leaf index pages with one row per page at the leaf level. You can use this to reduce lock contention on frequently accessed data.

Low **max_rows_per_page** values cause Adaptive Server to create new indexes with pages that are not completely full, uses more storage space, and may cause more page splits.

◆ ***WARNING!***

Creating a clustered index with max_rows_per_page can affect the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

on *segment_name* – specifies that the index is to be created on the named segment. Before the **on** *segment_name* option can be used, the device must be initialized with **disk init**, and the segment must be added to the database with the **sp_addsegment** system procedure. See your System Administrator or use **sp_helpsegment** for a list of the segment names available in your database.

If you specify **clustered** and use the **on** *segment_name* option, the entire table migrates to the segment you specify, since the leaf level of the index contains the actual data pages.

references – specifies a column list for a referential integrity constraint. You can specify only one column value for a column-constraint. By including this constraint with a table that references another table, any data inserted into the **referencing** table must already exist in the **referenced** table.

To use this constraint, you must have **references** permission on the referenced table. The specified columns in the referenced table must be constrained by a unique index (created by either a **unique** constraint or a **create index** statement). If no columns are specified, there must be a **primary key** constraint on the appropriate columns in the referenced table. Also, the datatypes of the referencing table columns must exactly match the datatype of the referenced table columns.

If Component Integration Services is enabled, you cannot use **references** for remote servers.

foreign key – specifies that the listed column(s) are foreign keys in this table whose matching primary keys are the columns listed in the **references** clause.

ref_table – is the name of the table that contains the referenced columns. You can reference tables in another database. Constraints can reference up to 192 user tables and internally generated worktables. Use the system procedure **sp_helpconstraint** to check a table's referential constraints.

ref_column – is the name of the column or columns in the referenced table.

check – specifies a *search_condition* constraint that Adaptive Server enforces for all the rows in the table.

If Component Integration Services is enabled, you cannot use **check** for remote servers.

search_condition – is a boolean expression that defines the **check** constraint on the column values. These constraints can include:

- A list of constant expressions introduced with **in**.
- A set of conditions, which may contain wildcard characters, introduced with **like**.

An expression can include arithmetic operations and Transact-SQL functions. The *search_condition* cannot contain subqueries, aggregate functions, parameters, or host variables.

next_column – includes additional column definitions (separated by commas) using the same syntax described for a column definition.

drop – specifies the name of a column or constraint to drop from the table.

If Component Integration Services is enabled, you cannot use **drop** for remote servers.

modify – specifies the name of the column whose datatype or nullability you are changing.

replace – specifies the column whose default value you want to change with the new value specified by a following **default** clause.

If Component Integration Services is enabled, you cannot use **replace** for remote servers.

partition *number_of_partitions* – creates multiple database page chains for the table. Adaptive Server can perform concurrent insertion operations into the last page of each chain. *number_of_partitions* must be a positive integer greater than or equal to 2. Each partition requires an additional control page; lack of disk space can limit the number of partitions you can create in a table. Lack of memory can limit the number of partitioned tables you can access.

If Component Integration Services is enabled, you cannot use **partition** for remote servers.

unpartition – creates a single page chain for the table by concatenating subsequent page chains with the first one.

If Component Integration Services is enabled, you cannot use **unpartition** for remote servers.

asc | desc – specifies whether the index is to be created in ascending (**asc**) or descending (**desc**) order. The default is ascending order.

reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations for the index created by the constraint. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value, 0, leaves no empty pages.

lock datarows | datapages | allpages – changes the locking scheme to be used for the table.

exp_row_size = *num_bytes* – specifies the expected row size; applies only to **datarows** and **datapages** locking schemes, to tables with variable-length rows, and only when **alter table** performs a data copy. Valid values are 0, 1, and any value between the minimum and maximum row length for the table. The default value is 0, which means a server-wide setting is applied.

Examples

1.
```
alter table publishers
add manager_name varchar(40) null
```

   Adds a column to a table. For each existing row in the table, Adaptive Server assigns a NULL column value.

2.
```
alter table sales_daily
add ord_num numeric(5,0) identity
```

   Adds an IDENTITY column to a table. For each existing row in the table, Adaptive Server assigns a unique, sequential column

value. Note that the IDENTITY column has type *numeric* and a scale of zero. The precision determines the maximum value (10^5 - 1, or 99,999) that can be inserted into the column.

3.
```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
```

   Adds a primary key constraint to the *authors* table. If there is an existing primary key or unique constraint on the table, the existing constraint must be dropped first (see example 5).

4.
```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
with reservepagegap = 16
```

   Creates an index on *authors*; the index has a **reservepagegap** value of 16, leaving 1 empty page in the index for each 15 allocated pages.

5.
```
alter table titles
drop constraint au_identification
```

   Drops the *au_identification* constraint.

6.
```
alter table authors
replace phone default null
```

   Removes the default constraint on the *phone* column in the *authors* table. If the column allows NULL values, NULL is inserted if no column value is specified. If the column does not allow NULL values, an insert that does not specify a column value fails.

7.
```
alter table titleauthor partition 5
```

   Creates four new page chains for the *titleauthor* table. After the table is partitioned, existing data remains in the first partition. New rows, however, are inserted into all five partitions.

8.
```
alter table titleauthor unpartition
alter table titleauthor partition 6
```

   Concatenates all page chains of the *titleauthor* table, then repartitions it with six partitions.

9.
```
alter table titles lock datarows
```

   Changes the locking scheme for the *titles* table to datarows locking.

```
10.alter table authors
       add author_type varchar(20)
       default "primary_author" not null
```

Adds the not-null column *author_type* to the *authors* table with a default of *primary_author.*

```
11.alter table titles
       drop advance, notes, contract
```

Drops the *advance*, *notes*, and *contract* columns from the *titles* table.

```
12.alter table authors
       modify city varchar(30) null
```

Modifies the *city* column of the *authors* table to be a *varchar(30)* with a default of NULL.

```
13.alter table stores
       modify stor_name not null
```

Modifies the *stor_name* column of the *stores* table to be NOT NULL. Note that its datatype, *varchar(40)*, remains unchanged.

```
14.alter table titles
       modify type varchar(10)
       lock datarows
```

Modifies the *type* column of the *titles* table and changes the locking scheme of the *titles* table from allpages to datarows.

```
15.alter table titles
       modify notes varchar(150) not null
       with exp_row_size = 40
```

Modifies the *notes* column of the *titles* table from *varchar(200)* to *varchar(150)*, changes the default value from NULL to NOT NULL, and specifies an **exp_row_size** of 40.

```
16.alter table titles
       add author_type varchar(30) null
       modify city varchar(30)
       drop notes
       add sec_advance money default 1000 not null
       lock datarows
       with exp_row_size = 40
```

Adds, modifies, and drops a column, and then adds another column in one query. Alters the locking scheme and specifies the **exp_row_size** of the new column.

Comments

- If stored procedures using **select *** reference a table that has been altered, no new columns appear in the result set, even if you use the **with recompile** option. You must drop the procedure and re-create it to include these new columns.

Restrictions

- You cannot add a column of datatype *bit* to an existing table.
- The number of columns in a table cannot exceed 250. The maximum number of bytes per row depends on the locking scheme for the table. The maximum number of bytes for user data is 1960 bytes in an allpages-locked table. For data-only-locked tables, deduct 2 bytes for each variable-length column or column that allows null values.

◆ ***WARNING!***

Do not alter the system tables.

- You cannot partition a system table or a table that is already partitioned.
- You cannot issue the **alter table command** with a **partition** or **unpartition** clause within a user-defined transaction.

Getting Information About Tables

- For information about a table and its columns, use **sp_help**.
- To rename a table, execute the system procedure **sp_rename** (do not rename the system tables).
- For information about integrity constraints (**unique**, **primary key**, **references**, and **check**) or the **default** clause, see **create table** in this chapter.

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing. For more information, see “Indexing for Performance” in the *Performance and Tuning Guide*.

Using Cross-Database Referential Integrity Constraints

- When you create a cross-database constraint, Adaptive Server stores the following information in the *sysreferences* system table of each database:

Table 6-2: Information stored about referential integrity constraints

Information Stored in *sysreferences*	Columns with Information About the Referenced Table	Columns with Information About the Referencing Table
Key column IDs	*refkey1* through *refkey16*	*fokey1* through *fokey16*
Table ID	*reftabid*	*tableid*
Database ID	*pmrydbid*	*frgndbid*
Database name	*pmrydbname*	*frgndbname*

- When you drop a referencing table or its database, Adaptive Server removes the foreign key information from the referenced database.
- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains the referenced table, or
  - Rename either database with **sp_renamedb**.

  You must first remove the cross-database constraint with **alter table.**
- Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of these databases could cause database corruption.

- The *sysreferences* system table stores the **name** and the ID number of the external database. Adaptive Server cannot guarantee referential integrity if you use **load database** to change the database name or to load it onto a different server.

◆ ***WARNING!***

Before dumping a database in order to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

Changing Defaults

- You can create column defaults in two ways: by declaring the default as a column constraint in the **create table** or **alter table** statement or by creating the default using the **create default** statement and binding it to a column using **sp_bindefault**.
- You cannot replace a user-defined default bound to the column with **sp_bindefault**. Unbind the default with **sp_unbindefault** first.
- If you declare a default column value with **create table** or **alter table**, you cannot bind a default to that column with **sp_bindefault**. Drop the default by altering it to NULL, then bind the user-defined default. Changing the default to NULL unbinds the default and deletes it from the *sysobjects* table.

Setting Space Management Properties for Indexes

- The space management properties **fillfactor**, **max_rows_per_page**, and **reservepagegap** in the **alter table** statement apply to indexes that are created for **primary key** or **unique** constraints. The space management properties affect the data pages of the table if the constraint creates a clustered index on an allpages-locked table.
- Use **sp_chgattribute to** change **max_rows_per_page** or **reservepagegap** for a table or an index, or to store **fillfactor** values.
- Space management properties for indexes are applied:
  - When indexes are re-created as a result of an **alter table** command that changes the locking scheme for a table from allpages locking to data-only locking or vice versa. See “Changing Locking Schemes” on page -36 for more information.
  - When indexes are automatically rebuilt as part of a **reorg rebuild** command.
- To see the space management properties currently in effect for a table, use **sp_help**. To see the space management properties currently in effect for an index, use **sp_helpindex**.

- The space management properties **fillfactor**, **max_rows_per_page**, and **reservepagegap** help manage space usage for tables and indexes in the following ways:
  - **fillfactor** leaves extra space on pages when indexes are created, but the **fillfactor** is not maintained over time. It applies to all locking schemes.
  - **max_rows_per_page** limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables.
  - **reservepagegap** specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to all locking schemes.

  Space management properties can be stored for tables and indexes so that they are applied during alter table and reorg rebuild commands.
- Table 6-3 shows the valid combinations of space management properties and locking schemes. If an **alter table** command changes the table so that the combination is not compatible, the values stored in the stored in system tables remain there, but are not applied during operations on the table. If the locking scheme for a table changes so that the properties become valid, then they are used.

Table 6-3: Space management properties and locking schemes

Parameter	*allpages*	*datapages*	*datarows*
max_rows_per_page	Yes	No	No
reservepagegap	Yes	Yes	Yes
fillfactor	Yes	Yes	Yes
exp_row_size	No	Yes	Yes

- Table 6-4 shows the default values and the effects of using the default values for the space management properties.

Table 6-4: Defaults and effects of space management properties

Parameter	Default	Effect of Using the Default
max_rows_per_page	0	Fits as many rows as possible on the page, up to a maximum of 255

Table 6-4: Defaults and effects of space management properties

Parameter	Default	Effect of Using the Default
reservepagegap	0	Leaves no gaps
fillfactor	0	Fully packs leaf pages

Conversion of *max_rows_per_page* to *exp_row_size*

- If a table has **max_rows_per_page** set, and the table is converted from allpages locking to data-only locking, the value is converted to an **exp_row_size** value before the **alter table...lock** command copies the table to its new location. The **exp_row_size** is enforced during the copy. Table 6-5 shows how the values are converted.

Table 6-5: Converting max_rows_per_page to exp_row_size

If *max_rows_per_page* is set to	Set *exp_row_size* to
0	Percentage value set by **default exp_row_size percent**
255	1, that is, fully packed pages
1–254	The smaller of: • maximum row size • 2002 / **max_rows_per_page** value

Using *reservepagegap*

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The **reservepagegap** keyword causes these commands to leave empty pages so that future page allocations take place close to the page that is being split or to the page from which a row is being forwarded.
- The **reservepagegap** value for a table is stored in *sysindexes*, and is applied when the locking scheme for a table is changed from allpages locking to data-only locking or vice versa. To change the stored value, use the system procedure **sp_chgattribute** before running **alter table**.
- **reservepagegap** specified with the **clustered** keyword on an allpages-locked table overwrites any value previously specified with **create table** or **alter table**.

Partitioning Tables for Improved Insert Performance

- Partitioning a table with the **partition** clause of the **alter table** command creates additional page chains, making multiple last pages available at any given time for concurrent insert operations. This improves insert performance by reducing page contention and, if the segment containing the table is spread over multiple physical devices, by reducing I/O contention while the server flushes data from cache to disk.
- If you are copying data into or out of a partitioned table, the Adaptive Server must be configured for parallel processing.
- When you partition a table, Adaptive Server allocates a control page for each partition, including the first partition. The existing page chain becomes part of the first partition. Adaptive Server creates a first page for each subsequent partition. Since each partition has its own control page, partitioned tables require slightly more disk space than unpartitioned tables.
- You can partition both empty tables and those that contain data. Partitioning a table does **not** move data; existing data remains where it was originally stored, in the first partition. For best performance, partition a table **before** inserting data.
- You cannot partition a system table or a table that is already partitioned. You can partition a table that contains *text* and *image* columns; however, partitioning has no effect on the way Adaptive Server stores the *text* and *image* columns.
- After you have partitioned a table, you cannot use the **truncate table** command or the **sp_placeobject** system procedure on it.
- To change the number of partitions in a table, use the **unpartition** clause of **alter table** to concatenate all existing page chains, then use the **partition** clause of **alter table** to repartition the table.
- If you unpartition a table, recompile the query plans of any dependent procedures. Unpartitioning does not automatically recompile procedures.
- When you unpartition a table with the **unpartition** clause of the **alter table** command, Adaptive Server deallocates all control pages, including that of the first partition, and concatenates the page chains. The resulting single page chain contains no empty pages, with the possible exception of the first page. Unpartitioning a table does **not** move data.

Adding IDENTITY Columns

- When adding an IDENTITY column to a table, make sure the column precision is large enough to accommodate the number of existing rows. If the number of rows exceeds 10 $^{\text{PRECISION}}$ - 1, Adaptive Server prints an error message and does not add the column.
- When adding an IDENTITY column to a table, Adaptive Server:
  - Locks the table until all the IDENTITY column values have been generated. If a table contains a large number of rows, this process may be time-consuming.
  - Assigns each existing row a unique, sequential IDENTITY column value, beginning with the value 1.
  - Logs each insert operation into the table. Use **dump transaction** to clear the database's transaction log before adding an IDENTITY column to a table with a large number of rows.
- Each time you insert a row into the table, Adaptive Server generates an IDENTITY column value that is one higher than the last value. This value takes precedence over any defaults declared for the column in the **alter table** statement or bound to it with **sp_bindefault**.

Altering Table Schema

- **add**, **drop**, or **modify,** and **lock** sub-clauses are useful to change an existing table's schema. A single statement can contain any number of these sub-clauses, in any order, as long as the same column name is not referenced more than once in the statement.
- If stored procedures using **select *** reference a table that has been altered, no new columns appear in the result set, even if you use the **with recompile** option. You must drop the procedure and re-create it to include these new columns.
- You cannot drop all the columns in a table. Also, you cannot drop the last remaining column from a table (for example, if you drop four columns from a five-column table, you cannot then drop the remaining column). To remove a table from the database, use **drop table**.
- Data copy is required:
  - To drop a column
  - To add a NOT NULL column

- For most **alter table. . . modify** commands

  Use **showplan** to determine if a data copy is required for a particular **alter table** command.

- You can specify a change in the locking scheme for the modified table with other **alter table** commands (**add**, **drop**, or **modify**) when the other **alter table** command requires a data copy.
- If **alter table** performs a data copy, **select into /bulkcopy/pllsort** must be turned on in the database that includes the table whose schema you are changing.
- Adaptive Server must be configured for parallel processing when you alter the schema of a partitioned table and the change requires a data copy.
- The modified table retains the existing space management properties (**max_rows_per_page**, **fillfactor**, and so on) and indexes of the table.
- **alter table** that requires a data copy does not fire any triggers.
- You can use **alter table** to change the schema of remote proxy tables created and maintained by Component Integration Services (CIS). For information about CIS, see the *Component Integration Services User's Guide*.
- You cannot perform a data copy and add a table level or referential integrity constraint in the same statement.
- You cannot perform a data copy and create a clustered index in the same statement.
- If you add a NOT NULL column, you must also specify a default clause.
- You can always add, drop, or modify a column in an all-pages locked tables. However, there are restrictions for adding, dropping, or modifying a column in a data-only locked table, which are described in Table 6-6:

Table 6-6: When can I add, drop, or modify a column in data-only locked table?

Type of index	All -pages Locking, partitioned table	All-pages Locked, unpartitioned table	Datat-only locked, partitioned table	Datat-only locked, partitioned table
Clustered	Yes	Yes	No	Yes

Table 6-6: When can I add, drop, or modify a column in data-only locked table?

Type of index	All -pages Locking, partitioned table	All-pages Locked, unpartitioned table	Datat-only locked, partitioned table	Datat-only locked, partitioned table
Non-clustered	Yes	Yes	Yes	Yes

If you need to add, drop, or modify a column in a data-only locked table partitioned table with a clustered index, you can:

1. Drop the clustered index.
2. Alter the (data-only locked) table.
3. Re-create the clustered index.

- You cannot add a NOT NULL Java object as a column. By default, all Java columns always have a default value of NULL, and are stored as either *varbinary* strings or as *image* datatypes.
- You cannot modify a partitioned table that contains a Java column if the modification requires a data copy. Instead, first unpartition the table, run the **alter table** command, then repartition the table.
- You cannot drop the key column from an index or a referential integrity constraint. To drop a key column, first drop the index or referential integrity constraint, then drop the key column. See the *Transact-SQL User's Guide* for more information.
- You cannot drop columns that contain or are referenced by rules, constraints, or defaults. Instead, first drop the rule, constraint, or default, then drop the column. Use **sp_helpconstraint** to identify any constraints on a table, and use **sp_depends** to identify any column-level dependencies.
- You cannot drop a column from a system table. Also, you cannot drop columns from user tables that are created and used by Sybase-provided tools and stored procedures.
- You can generally modify the datatype of an existing column to any other datatype if the table is empty. If the table is not empty, you can modify the datatype to any datatype that is explicitly convertible to the original datatype.
- You can:
  - Add a new IDENTITY column.

- Drop an existing IDENTITY column.
- Modify the size of an existing IDENTITY.

See the *Transact-SQL User's Guide* for more information.

- Altering the schema of a table increments the schema count, causing existing stored procedures that access this table to be renormalized the next time they are executed. Changes in datatype-dependent stored procedures or views may fail with datatype normalization type errors. You must update these dependent objects so they refer to the modified schema of the table.

Restrictions for Modifying A Table Schema

- You cannot run **alter table** from inside a transaction.
- Altering a table's schema can invalidate backups that you made using **bcp**. These backups may use a tables schema that is no longer compatible with the table's current schema.
- You can add NOT NULL columns with check constraints, however, Adaptive Server does not validate the constraint against existing data.
- You cannot change the locking scheme of a table using the **alter table . . . add**, **drop**, or **modify** commands if the table has a clustered index and the operation requires a data copy. Instead you can

  1. Drop the clustered index.
  2. Alter the table's schema.
  3. Re-create the clustered index.

- You cannot alter a table's schema if there are any active open cursors on the table.

Restrictions for Modifying *text* And *image* Columns

- You can only add *text* or *image* columns that accept null values.

  To add a *text* or *image* column so it contains only non-null values, first add a column that only accepts null values and then update it to the non-null values.
- You can only modify a column from *text* datatype to the following datatypes:
  - *char*
  - *varchar*

  - *nchar*
  - *nvarchar*
- You can only modify a column from *image* datatype to a *varbinary* datatype, and the column can only include non-null data.
- You can modify *text* or *image* columns to any other datatypes only if the table is empty.
- You cannot add a new *text* or *image* column and then drop an existing *text* or *image* column in the same statement.
- You cannot modify a column to either *text* or *image* datatype.

Changing Locking Schemes

- **alter table** supports changing from any locking scheme to any other locking scheme. You can change:
  - From **allpages** to **datapages** or vice versa
  - From **allpages** to **datarows** or vice versa
  - From **datapages** to **datarows** or vice versa
- Before you change from allpages locking to a data-only locking scheme, or vice versa, use **sp_dboption** to set the database option **select into/bulkcopy/pllsort** to true, then run **checkpoint** in the database if any of the tables are partitioned and the sorts for the indexes require a parallel sort.
- After changing the locking scheme from allpages-locking to data-only locking or vice versa, the use of the **dump transaction** command to back up the transaction log is prohibited; you must first perform a full database dump.
- When you use **alter table...lock** to change the locking scheme for a table from allpages locking to data-only locking or vice versa, Adaptive Server makes a copy of the table's data pages. There must be enough room on the segment where the table resides for a complete copy of the data pages. There must be space on the segment where the indexes reside to rebuild the indexes.

  Clustered indexes for data-only-locked tables have a leaf level above the data pages. If you are altering a table with a clustered index from allpages-locking to a data-only-locking, the resulting clustered index requires more space. The additional space required depends on the size of the index keys.

Use **sp_spaceused** to determine how much space is currently occupied by the table, and use **sp_helpsegment** to see the space available to store the table.

- When you change the locking scheme for a table from allpages locking to datapages locking or vice versa, the space management properties are applied to the tables, as the data rows are copied, and to the indexes, as they are re-created. When you change from one data-only locking scheme to another, the data pages are not copied, and the space management properties are not applied.
- If a table is partitioned, changing the locking scheme performs a partition-to-partition copy of the rows. It does not balance the data on the partitions during the copy.
- When you change the locking scheme for a table, the **alter table...lock** command acquires an exclusive lock on the table until the command completes.
- When you use **alter table...lock** to change from datapages locking to datarows locking, the command does not copy data pages or rebuild indexes. It only updates system tables.
- Changing the locking scheme while other users are active on the system may have the following effects on user activity:
  - Query plans in the procedure cache that access the table will be recompiled the next time they are run.
  - Active multi-statement procedures that use the table are recompiled before continuing with the next step.
  - Ad hoc batch transactions that use the table are terminated.

◆ ***WARNING!***

Changing the locking scheme for a table while a bulk copy operation is active can cause table corruption. Bulk copy operates by first obtaining information about the table and does not hold a lock between the time it reads the table information and the time it starts sending rows, leaving a small window of time for an alter table...lock command to start.

Adding Java-SQL Columns

- If Java is enabled in the database, you can add Java-SQL columns to a table. For more information, see *Java in Adaptive Server Enterprise*.
- The declared class (*datatype*) of the new Java-SQL column must implement either the **Serializable** or **Externalizable** interface.
- When you add a Java-SQL column to a table, the Java-SQL column cannot be specified:
  - As a foreign key
  - In a references clause
  - As having the UNIQUE property
  - As the primary key
- If **in row** is specified, then the value stored cannot exceed 255 bytes.
- If **off row** is specified, then:
  - The column cannot be referenced in a check constraint.
  - The column cannot be referenced in a **select** that specifies **distinct.**
  - The column cannot be specified in a comparison operator, in a predicate, or in a **group by** clause.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Transact-SQL extension.	See Chapter 1, "System and User-Defined Datatypes" for datatype compliance information.

Permissions

alter table permission defaults to the table owner; it cannot be transferred except to the Database Owner, who can impersonate the table owner by running the **setuser** command. A System Administrator can also alter user tables.

See Also

Commands	**create index**, **create table**, **dbcc**, **drop database**, **insert**
System procedures	**sp_chgattribute**, **sp_help**, **sp_helpartition**, **sp_rename**

begin...end

Function

Encloses a series of SQL statements so that control-of-flow language, such as **if...else**, can affect the performance of the whole group.

Syntax

```
begin
   statement block
end
```

Keywords and Options

statement block – is a series of statements enclosed by **begin** and **end**.

Examples

1.
```
if (select avg(price) from titles) < $15
begin
  update titles
  set price = price * $2
  select title, price
  from titles
  where price > $28
end
```

Without **begin** and **end**, the **if** condition would cause execution of only one SQL statement.

2.
```
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
 where salesdetail.title_id = deleted.title_id) > 0
  begin
    rollback transaction
    print "You can't delete a title with sales."
  end
else
  print "Deletion successful--no sales for this
      title."
```

Without **begin** and **end**, the **print** statement would not execute.

Comments

- **begin...end** blocks can nest within other **begin...end** blocks.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

begin...end permission defaults to all users. No permission is required to use it.

See Also

Commands	if...else

begin transaction

Function

Marks the starting point of a user-defined transaction.

Syntax

```
begin tran[saction] [transaction_name]
```

Keywords and Options

transaction_name – is the name assigned to this transaction. Transaction names must conform to the rules for identifiers. Use transaction names only on the outermost pair of nested **begin transaction** / **commit** or **begin transaction** / **rollback** statements.

Examples

```
1. begin transaction
     insert into publishers (pub_id) values ("9999")
   commit transaction
```

Explicitly begins a transaction for the **insert** statement.

Comments

- Define a transaction by enclosing SQL statements and / or system procedures within the phrases **begin transaction** and **commit**. If you set chained transaction mode, Adaptive Server implicitly invokes a **begin transaction** before the following statements: **delete**, **insert**, **open**, **fetch**, **select**, and **update**. You must still explicitly close the transaction with a **commit**.
- To cancel all or part of a transaction, use the **rollback** command. The **rollback** command must appear within a transaction; you cannot roll back a transaction after it is committed.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

begin transaction permission defaults to all users. No permission is required to use it.

See Also

Commands	**commit**, **rollback**, **save transaction**

break

Function

Causes an exit from a **while** loop. **break** is often activated by an **if** test.

Syntax

```
while logical_expression
      statement
   break
      statement
   continue
```

Keywords and Options

logical_expression – is an expression (a column name, constant, any combination of column names and constants connected by arithmetic or bitwise operators, or a subquery) that returns TRUE, FALSE, or NULL. If the logical expression contains a **select** statement, enclose the **select** statement in parentheses.

Examples

```
1. while (select avg(price) from titles) < $30
    begin
     update titles
     set price = price * 2
     select max(price) from titles
     if (select max(price) from titles) > $50
        break
     else
        continue
    end
    begin
     print "Too much for the market to bear"
    end
```

If the average price is less than $30, double the prices. Then, select the maximum price. If it is less than or equal to $50, restart the **while** loop and double the prices again. If the maximum price is more than $50, exit the **while** loop and print a message.

Comments

- **break** causes an exit from a **while** loop. Statements that appear after the keyword **end**, which marks the end of the loop, are then executed.

- If two or more **while** loops are nested, the inner **break** exits to the next outermost loop. First, all the statements after the end of the inner loop run; then, the next outermost loop restarts.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

break permission defaults to all users. No permission is required to use it.

See Also

Commands	**continue**, **while**

case

Function

Supports conditional SQL expressions; can be used anywhere a value expression can be used.

Syntax

```
case
    when search_condition then expression
    [when search_condition then expression]...
    [else expression]
end
```

case and values syntax:

```
case expression
    when expression then expression
    [when expression then expression]...
    [else expression]
end
```

Keywords and Options

case – begins the **case** expression.

when – precedes the search condition or the expression to be compared.

search_condition – is used to set conditions for the results that are selected. Search conditions for **case** expressions are similar to the search conditions in a **where** clause. Search conditions are detailed in the *Transact-SQL User's Guide*.

then – precedes the expression that specifies a result value of **case**.

expression – is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

1.
```
select au_lname, postalcode,
    case
        when postalcode = "94705"
            then "Berkeley Author"
        when postalcode = "94609"
            then "Oakland Author"
        when postalcode = "94612"
            then "Oakland Author"
        when postalcode = "97330"
            then "Corvallis Author"
    end
from authors
```

   Selects all the authors from the *authors* table and, for certain authors, specifies the city in which they live.

2.
```
select stor_id, discount,
    coalesce (lowqty, highqty)
from discounts
```

   Returns the first occurrence of a non-NULL value in either the *lowqty* or *highqty* column of the *discounts* table:

3.
```
select stor_id, discount,
    case
        when lowqty is not NULL then lowqty
        else highqty
    end
from discounts
```

   This is an alternative way of writing example 2.

4.
```
select title,
    nullif(type, "UNDECIDED")
from titles
```

   Selects the *titles* and *type* from the *titles* table. If the book type is UNDECIDED, **nullif** returns a NULL value.

5.
```
select title,
    case
        when type = "UNDECIDED" then NULL
        else type
    end
from titles
```

   This is an alternative way of writing example 4.

Comments

- **case** expression simplifies standard SQL expressions by allowing you to express a search condition using a **when...then** construct instead of an **if** statement.
- **case** expressions can be used anywhere an expression can be used in SQL.
- At least one result of the **case** expression must return a non-null value. For example:

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

  results in the following error message:

```
All result expressions in a CASE expression must not be NULL.
```

- If your query produces a variety of datatypes, the datatype of a **case** expression result is determined by datatype hierarchy, as described in "Datatype of Mixed-Mode Expressions" in Chapter 1, "System and User-Defined Datatypes." If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, *char* and *int*), the query fails.
- **coalesce** is an abbreviated form of a **case** expression. Example 3 describes an alternative way of writing the **coalesce** statement.
- **coalesce** must be followed by at least two expressions. For example:

```
select stor_id, discount,
    coalesce (highqty)
from discounts
```

  results in the following error message:

```
A single coalesce element is illegal in a COALESCE expression.
```

- **nullif** is an abbreviated form of a **case** expression. Example 5 describes an alternative way of writing **nullif**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

case permission defaults to all users. No permission is required to use it.

See Also

Commands	select, if...else, where Clause

checkpoint

Function

Writes all **dirty** pages (pages that have been updated since they were last written) to the database device.

Syntax

```
checkpoint
```

Examples

1. `checkpoint`

Writes all dirty pages in the current database to the database device, regardless of the system checkpoint schedule.

Comments

- Use **checkpoint** only as a precautionary measure in special circumstances. For example, Adaptive Server instructs you to issue the **checkpoint** command after resetting database options.
- Use **checkpoint** each time you change a database option with the system procedure **sp_dboption**.

Automatic Checkpoints

- Checkpoints caused by the **checkpoint** command supplement automatic checkpoints, which occur at intervals calculated by Adaptive Server on the basis of the configurable value for maximum acceptable recovery time.
- The **checkpoint** shortens the automatic recovery process by identifying a point at which all completed transactions are guaranteed to have been written to the database device. A typical **checkpoint** takes about 1 second, although checkpoint time varies, depending on the amount of activity on Adaptive Server.
- The automatic **checkpoint** interval is calculated by Adaptive Server on the basis of system activity and the recovery interval value in the system table *syscurconfigs*. The recovery interval determines **checkpoint** frequency by specifying the maximum amount of time it should take for the system to recover. Reset this value by executing the system procedure **sp_configure**.
- If the housekeeper task is able to flush all active buffer pools in all configured caches during the server's idle time, it wakes up the

checkpoint task. The checkpoint task determines whether it can checkpoint the database.

Checkpoints that occur as a result of the housekeeper task are known as **free checkpoints**. They do not involve writing many dirty pages to the database device, since the housekeeper task has already done this work. They may improve recovery speed for the database.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

checkpoint permission defaults to the Database Owner. It cannot be transferred.

See Also

System procedures	**sp_configure**, **sp_dboption**

close

Function

Deactivates a cursor.

Syntax

```
close cursor_name
```

Parameters

cursor_name – is the name of the cursor to close.

Examples

```
1. close authors_crsr
```

Closes the cursor named *authors_crsr*.

Comments

- The **close** command essentially removes the cursor's result set. The cursor position within the result set is undefined for a closed cursor.
- Adaptive Server returns an error message if the cursor is already closed or does not exist.

Standards and Compliance

Standard	Compliance Level
SQL92	Entry level compliant

Permissions

close permission defaults to all users. No permission is required to use it.

See Also

Commands	**deallocate cursor**, **declare cursor**, **fetch**, **open**

coalesce

Function

Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a **case** expression.

Syntax

```
coalesce(expression, expression [, expression]...)
```

Keywords and Options

coalesce – evaluates the listed expressions and returns the first non-null value. If all the expressions are null, **coalesce** returns a null.

expression – is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

```
1. select stor_id, discount,
       coalesce (lowqty, highqty)
   from discounts
```

Returns the first occurrence of a non-NULL value in either the *lowqty* or *highqty* column of the *discounts* table:

```
2. select stor_id, discount,
       case
           when lowqty is not NULL then lowqty
           else highqty
       end
   from discounts
```

This is an alternative way of writing example 1.

Comments

- **coalesce** expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a **when...then** construct.
- **coalesce** expressions can be used anywhere an expression can be used in SQL.

- At least one result of the **coalesce** expression must return a non-null value. For example:

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

  results in the following error message:

```
All result expressions in a CASE expression must not be NULL.
```

- If your query produces a variety of datatypes, the datatype of a **case** expression result is determined by datatype hierarchy, as described in "Datatype of Mixed-Mode Expressions" in Chapter 1, "System and User-Defined Datatypes." If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, *char* and *int*), the query fails.
- **coalesce** is an abbreviated form of a **case** expression. Example 2 describes an alternative way of writing the **coalesce** statement.
- **coalesce** must be followed by at least two expressions. For example:

```
select stor_id, discount,
    coalesce (highqty)
from discounts
```

  results in the following error message:

```
A single coalesce element is illegal in a COALESCE expression.
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

coalesce permission defaults to all users. No permission is required to use it.

See Also

Commands	**case**, **nullif**, **select**, **if...else**, **where Clause**

commit

Function

Marks the ending point of a user-defined transaction.

Syntax

```
commit [tran[saction] | work] [transaction_name]
```

Keywords and Options

transaction | **tran** | **work** – is optional.

transaction_name – is the name assigned to the transaction. It must conform to the rules for identifiers. Use transaction names only on the outermost pair of nested **begin transaction** / **commit** or **begin transaction** / **rollback** statements.

Examples

```
1. begin transaction royalty_change

   update titleauthor
   set royaltyper = 65
   from titleauthor, titles
   where royaltyper = 75
   and titleauthor.title_id = titles.title_id
   and title = "The Gourmet Microwave"

   update titleauthor
   set royaltyper = 35
   from titleauthor, titles
   where royaltyper = 25
   and titleauthor.title_id = titles.title_id
   and title = "The Gourmet Microwave"

   save transaction percentchanged

   update titles
   set price = price * 1.1
```

```
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

rollback transaction percentchanged

commit transaction
```

After updating the *royaltyper* entries for the two authors, insert the savepoint *percentchanged*, then determine how a 10 percent increase in the book's price would affect the authors' royalty earnings. The transaction is rolled back to the savepoint with the **rollback transaction** command.

Comments

- Define a transaction by enclosing SQL statements and / or system procedures with the phrases **begin transaction** and **commit.** If you set the chained transaction mode, Adaptive Server implicitly invokes a **begin transaction** before the following statements: **delete, insert, open, fetch, select,** and **update.** You must still explicitly enclose the transaction with a **commit.**
- To cancel all or part of an entire transaction, use the **rollback** command. The **rollback** command must appear within a transaction. You cannot roll back a transaction after the **commit** has been entered.
- If no transaction is currently active, the **commit** or **rollback** statement has no effect on Adaptive Server.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The **commit transaction** and **commit tran** forms of the statement are Transact-SQL extensions.

Permissions

commit permission defaults to all users.

See Also

Commands	**begin transaction**, **rollback**, **save transaction**

compute Clause

Function

Generates summary values that appear as additional rows in the query results.

Syntax

```
start_of_select_statement
   compute row_aggregate (column_name)
       [, row_aggregate(column_name)]...
   [by column_name [, column_name]...]
```

Keywords and Options

row_aggregate – is one of the following:

Function	Meaning
sum	Total of values in the (numeric) column
avg	Average of values in the (numeric) column
min	Lowest value in the column
max	Highest value in the column
count	Number of values in the column

column_name – is the name of a column. It must be enclosed in parentheses. Only numeric columns can be used with **sum** and **avg**.

One **compute** clause can apply several aggregate functions to the same set of grouping columns (see examples 2 and 3). To create more than one group, use more than one **compute** clause (see example 5).

by – calculates the row aggregate values for subgroups. Whenever the value of the **by** item changes, row aggregate values are generated. If you use **by**, you must use **order by**.

Listing more than one item after **by** breaks a group into subgroups and applies a function at each level of grouping.

Examples

1.
```
select type, price
from titles
where price > $12
  and type like "%cook"
  order by type, price
compute sum(price) by type
```

```
type       price
---------  ------------
mod_cook          19.99
           sum
           ------------
                  19.99
type       price
---------  ------------
trad_cook         14.99
trad_cook         20.95
           sum
           ------------
                  35.94
(5 rows affected)
```

   Calculates the sum of the prices of each type of cook book that costs more than $12.

2.
```
select type, price, advance
from titles
where price > $12
  and type like "%cook"
  order by type, price
compute sum(price), sum(advance) by type
```

```
type       price      advance
---------  ---------  ------------
mod_cook       19.99          0.00
          sum         sum
           ---------  ------------
               19.99          0.00

type       price      advance
---------  ---------  ------------
trad_cook      14.99      8,000.00
trad_cook      20.95      7,000.00
           sum        sum
           ---------  ------------
               35.94     15,000.00
(5 rows affected)
```

Calculates the sum of the prices and advances for each type of cook book that costs more than $12.

```
3. select type, price, advance
   from titles
   where price > $12
     and type like "%cook"
     order by type, price
   compute sum(price), max(advance) by type

   type      price     advance
   --------- --------- -------------
   mod_cook      19.99         0.00
             sum
             ---------
                 19.99
                       max
                       -------------
                                0.00

   type      price     advance
   --------- --------- -------------
   trad_cook     14.99      8,000.00
   trad_cook     20.95      7,000.00
             sum
             ---------
                 35.94
                        max
                        -------------
                             8,000.00
   (5 rows affected)
```

Calculates the sum of the prices and maximum advances of each type of cook book that costs more than $12.

```
4. select type, pub_id, price
   from titles
   where price > $10
     and type = "psychology"
     order by type, pub_id, price
   compute  sum(price) by type, pub_id

   type         pub_id    price
   ------------ --------- -----------
   psychology   0736          10.95
   psychology   0736          19.99
                            sum
                            ---------
                               30.94
```

```
type         pub_id    price
------------ --------- ---------
psychology   0877          21.59
                        sum
                        ---------
                            21.59
(5 rows affected)
```

Breaks on *type* and *pub_id* and calculates the sum of the prices of psychology books by a combination of type and publisher ID.

5.
```
select type, pub_id, price
from titles
where price > $10
  and type = "psychology"
order by type, pub_id, price
compute  sum(price) by type, pub_id
compute  sum(price) by type
```

```
type         pub_id    price
------------ --------- ---------
psychology   0736          10.95
psychology   0736          19.99
                        sum
                        ---------
                            30.94

type         pub_id    price
------------ --------- ---------
psychology   0877          21.59
                        sum
                        ---------
                            21.59
                        sum
                        ---------
                            52.53
(6 rows affected)
```

Calculates the grand total of the prices of psychology books that cost more than $10 in addition to calculating sums by *type* and *pub_id*.

6.
```
select type, price, advance
from titles
where price > $10
  and type like "%cook"
compute sum(price), sum(advance)
```

```
type       price        advance
---------  -----------  --------------
mod_cook         19.99            0.00
trad_cook        20.95        8,000.00
trad_cook        11.95        4,000.00
trad_cook        14.99        7,000.00
            sum          sum
           -----------  --------------
                 67.88       19,000.00
(5 rows affected)
```

Calculates the grand totals of the prices and advances of cook books that cost more than $10.

```
7. select type, price, price*2
   from titles
     where type like "%cook"
   compute sum(price), sum(price*2)
```

```
type         price
------------ --------------  ------------
mod_cook              19.99         39.98
mod_cook               2.99          5.98
trad_cook             20.95         41.90
trad_cook             11.95         23.90
trad_cook             14.99         29.98
             sum              sum
             =============    ============
                     70.87         141.74
```

Calculates the sum of the price of cook books and the sum of the price used in an expression.

Comments

- The **compute** clause allows you to see the detail and summary rows in one set of results. You can calculate summary values for subgroups, and you can calculate more than one aggregate for the same group.
- **compute** can be used without **by** to generate grand totals, grand counts, and so on. **order by** is optional if you use the **compute** keyword without **by**. See example 6.
- If you use **compute by**, you must also use an **order by** clause. The columns listed after **compute by** must be identical to or a subset of those listed after **order by** and must be in the same left-to-right order, start with the same expression, and not skip any expressions. For example, if the **order by** clause is:

  ```
  order by a, b, c
  ```

the **compute by** clause can be any (or all) of these:

```
compute by a, b, c
compute by a, b
compute by a
```

Restrictions

- You cannot use a **compute** clause in a cursor declaration.
- Summary values can be computed for both expressions and columns. Any expression or column that appears in the **compute** clause must appear in the **select** list.
- Aliases for column names are not allowed as arguments to the row aggregate in a **compute** clause, although they can be used in the **select** list, the **order by** clause, and the **by** clause of **compute**.
- You cannot use **select into** in the same statement as a **compute** clause, because statements that include **compute** do not generate normal tables.

compute Results Appear As a New Row or Rows

- The aggregate functions ordinarily produce a single value for all the selected rows in the table or for each group, and these summary values are shown as new columns. For example:

```
select type, sum(price), sum(advance)
from titles
where type like "%cook"
group by type
```

```
type
-------------  ---------  ----------
mod_cook           22.98  15,000.00
trad_cook          47.89  19,000.00

(2 rows affected)
```

- The **compute** clause makes it possible to retrieve detail and summary rows with one command. For example:

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
```

```
type        price       advance
----------  ----------  ----------------
mod_cook          2.99         15,000.00
mod_cook         19.99              0.00

Compute Result:
--------------------- ----------------
                 22.98         15,000.00
type         price         advance
----------  ----------  ----------------
trad_cook        11.95          4,000.00
trad_cook        14.99          8,000.00
trad_cook        20.95          7,000.00

Compute Result:
--------------------- ----------------
                 47.89         19,000.00
(7 rows affected)
```

- Table 6-7 lists the output and grouping of different types of **compute** clauses.

Table 6-7: compute by clauses and detail rows

Clauses and Grouping	Output	Examples
One **compute** clause, same function	One detail row	1, 2, 4, 6, 7
One **compute** clause, different functions	One detail row per type of function	3
More than one **compute** clause, same grouping columns	One detail row per **compute** clause; detail rows together in the output	Same results as having one compute clause with different functions
More than one **compute** clause, different grouping columns	One detail row per **compute** clause; detail rows in different places, depending on the grouping	5

Case Sensitivity

- If your server has a case-insensitive sort order installed, **compute** ignores the case of the data in the columns you specify. For example, given this data:

  ```
  select * from groupdemo
  ```

```
lname      amount
---------- ------------------
Smith                  10.00
smith                   5.00
SMITH                   7.00
Levi                    9.00
Lévi                   20.00
```

compute by on *lname* produces these results:

```
select lname, amount from groupdemo
order by lname
compute sum(amount) by lname
```

```
lname      amount
---------- -------------------------
Levi                            9.00

Compute Result:
-------------------------
                     9.00

lname      amount
---------- -------------------------
Lévi                           20.00

Compute Result:
-------------------------
                    20.00

lname      amount
---------- -------------------------
smith                           5.00
SMITH                           7.00
Smith                          10.00

Compute Result:
-------------------------
                    22.00
```

The same query on a case- and accent-insensitive server produces these results:

```
lname      amount
---------- ------------------------
Levi                          9.00
Lévi                         20.00

Compute Result:
------------------------
                     29.00

lname      amount
---------- ------------------------
smith                         5.00
SMITH                         7.00
Smith                        10.00

Compute Result:
------------------------
                     22.00
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

See Also

Commands	**group by and having Clauses**, **select**
Functions	**avg**, **count**, **max**, **min**, **sum**

connect to...disconnect

(Component Integration Services Only)

Function

Connects to the specified server and disconnects the connected server.

Syntax

```
connect to server_name
   disconnect
```

Keywords and Options

server_name – is the server to which a passthrough connection is required.

Examples

1. `connect to SYBASE`

Establishes a passthrough connection to the server named SYBASE.

2. `disconnect`

Disconnects the connected server.

Comments

- **connect to** specifies the server to which a passthrough connection is required. Passthrough mode enables you to perform native operations on a remote server.
- *server_name* must be the name of a server in the *sysservers* table, with its server class and network name defined.
- When establishing a connection to *server_name* on behalf of the user, Component Integration Services uses one of the following identifiers:
  - A remote login alias described in *sysattributes*, if present
  - The user's name and password

  In either case, if the connection cannot be made to the specified server, Adaptive Server returns an error message.
- For more information about adding remote servers, see **sp_addserver**.

- After making a passthrough connection, Component Integration Services bypasses the Transact-SQL parser and compiler when subsequent language text is received. It passes statements directly to the specified server, and converts the results into a form that can be recognized by the Open Client interface and returned to the client program.
- To close the connection created by the **connect to** command, use the **disconnect** command. You can use this command only after the connection has been made using **connect to**.
- The **disconnect** command can be abbreviated to **disc**.
- The **disconnect** command returns an error unless **connect to** has been previously issued and the server is connected to a remote server.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Permission to use the **connect to** command must be explicitly granted by the System Administrator. The syntax is:

```
grant connect to user_name
```

The System Administrator can grant or revoke connect permission to *public* globally while in the *master* database. If the System Administrator wants to grant or revoke **connect to** permission for a particular user, the user must be a valid user of the *master* database, and the System Administrator must first revoke permission from *public* as follows:

```
use master
go
revoke connect from public
go
sp_adduser fred
go
grant connect to fred
go
```

See Also

Commands	**create existing table**, **grant**

System Procedures	**sp_addserver**, **sp_autoconnect**, **sp_helpserver**, **sp_passthru**, **sp_remotesql**, **sp_serveroption**

continue

Function

Restarts the **while** loop. **continue** is often activated by an **if** test.

Syntax

```
while boolean_expression
      statement
   break
      statement
   continue
```

Examples

```
1. while (select avg(price) from titles) < $30
   begin
     update titles
     set price = price * 2
     select max(price) from titles

     if (select max(price) from titles) > $50
         break
     else
         continue
   end

   begin
   print "Too much for the market to bear"
   end
```

If the average price is less than $30, double the prices. Then, select the maximum price. If it is less than or equal to $50, restart the **while** loop and double the prices again. If the maximum price is more than $50, exit the **while** loop and print a message.

Comments

- **continue** restarts the **while** loop, skipping any statements after **continue**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

continue permission defaults to all users. No permission is required to use it.

See Also

Commands	**break**, **while**

create database

Function

Creates a new database.

Syntax

```
create database database_name
   [on {default | database_device} [= size]
       [, database_device [= size]]...]
   [log on database_device [= size]
       [, database_device [= size]]...]
   [with {override | default_location = "pathname"}]
   [for {load | proxy_update}]
```

Keywords and Options

database_name – is the name of the new database. It must conform to the rules for identifiers and cannot be a variable.

on – indicates a location and size for the database.

default – indicates that **create database** can put the new database on any default database device(s), as shown in *sysdevices.status*. To specify a size for the database without specifying a location, use this command:

```
on default = size
```

To change a database device's status to "default," use the system procedure **sp_diskdefault**.

database_device – is the logical name of the device on which to locate the database. A database can occupy different amounts of space on each of several database devices. Add database devices to Adaptive Server with **disk init**.

size – is the amount of space (in megabytes) allocated to the database. The Adaptive Server-supplied default size is 2MB.

log on – specifies the logical name of the device for the database logs. You can specify more than one device in the **log on** clause.

with override – forces Adaptive Server to accept your device specifications, even if they mix data and transaction logs on the same device, thereby endangering up-to-the-minute recoverability for your database. f you attempt to mix log and

data on the same device without using this clause, the **create database** command fails. If you mix log and data, and use **with override**, you are warned, but the command succeeds.

for load – invokes a streamlined version of **create database** that can be used only for loading a database dump. See "Using the for load Option", below, for more information.

with default_location – specifies the storage location of new tables. If the **for proxy_update** clause is also specified, one proxy table for each remote table or view is automatically created from the specified location.

for proxy_update – automatically gets metadata from the remote location and creates the proxy table. **for proxy_update** cannot be used unless **with default_location** is specified.

Examples

1. ```
   create database pubs
   ```

   Creates a database named *pubs*.

2. ```
   create database pubs
   on default = 4
   ```

   Creates a 4MB database named *pubs*.

3. ```
   create database pubs
   on datadev = 3, moredatadev = 2
   ```

   Creates a database named *pubs* with 3MB on the *datadev segment* and 2 MB on the *moredatadev* segment.

4. ```
   create database pubs
   on datadev = 3
   log on logdev = 1
   ```

   Creates a database named *pubs* with 3MB of data on the *datadev* segment and a 1MB log on the *logdev* segment.

5. ```
   create database proxydb
   with default_location
   "UNITEST.pubs.dbo."
   ```

   Creates a proxy database named *proxydb* but does not automatically create proxy tables.

6. ```
   create database proxydb
   on default = 4
   with default_location
   "UNITEST.pubs2.dbo."
   for proxy_update
   ```

Creates a proxy database named *proxydb* and automatically creates proxy tables.

Comments

- Use **create database** from the *master* database.
- If you do not specify a location and size for a database, the default location is any default database device(s) indicated in *master..sysdevices*. The default size is the larger of the size of the *model* database or the **default database size** parameter in *sysconfigures*.

  System Administrators can increase the default size by using **sp_configure** to change the value of **default database size** and restarting Adaptive Server. The **default database size** parameter must be at least as large as the *model* database. If you increase the size of the *model* database, the default size must also be increased.

  If Adaptive Server cannot give you as much space as you want where you have requested it, it comes as close as possible, on a per-device basis, and prints a message telling how much space was allocated and where it was allocated. The maximum size of a database is system-dependent.
- If a proxy database is created using:

  ```
  create database mydb on my_device
  with default_location = "pathname" for proxy_update
  ```

  The presence of the device name is enough to bypass size calculation, and this command may fail if the default database size (the size of the *model* database) isn't large enough to contain all of the proxy tables.

  To allow CIS to estimate database size, no device name or any other option should be provided with the command:

  ```
  create database mydb
  with default_location = "pathname" for proxy_update
  ```

Restrictions

- Adaptive Server can manage up to 32,767 databases.
- Adaptive Server can only create one database at a time. If two database creation requests collide, one user will get this message:

  ```
  model database in use: cannot create new database
  ```

- The maximum number of device fragments for a database is 128. Each time you allocate space on a database device with **create database** or **alter database**, that allocation represents a device fragment, and the allocation is entered as a row in *sysusages*.
- The maximum number of named segments for a database is 32. Segments are named subsets of database devices available to a particular Adaptive Server. For more information on segments, see the *System Administration Guide*.

New Databases Are Created from *model*

- Adaptive Server creates a new database by copying the *model* database.
- You can customize *model* by adding tables, stored procedures, user-defined datatypes, and other objects, and by changing database option settings. New databases inherit these objects and settings from *model*.
- To guarantee recoverability, the **create database** command must clear every page that was not initialized when the *model* database was copied. This may take several minutes, depending on the size of the database and the speed of your system.

  If you are creating a database in order to load a database dump into it, you can use the **for load** option to skip the page-clearing step. This makes database creation considerably faster.

Ensuring Database Recoverability

- Back up the *master* database each time you create a new database. This makes recovery easier and safer in case *master* is damaged.

➤ ***Note***

If you create a database and fail to back up *master*, you may be able to recover the changes with **disk refit**.

- The **with override** clause allows you to mix log and data segments on a single device. However, for full recoverability, the device or devices specified in **log on** should be different from the physical device that stores the data. In the event of a hard disk crash, the database can be recovered from database dumps and transaction logs.

A small database can be created on a single device that is used to store both the transaction log and the data, but you **must** rely on the **dump database** command for backups.

- The size of the device required for the transaction log varies according to the amount of update activity and the frequency of transaction log dumps. As a rule of thumb, allocate to the log device 10–25 percent of the space you allocate to the database itself. It is best to start small, since space allocated to a transaction log device cannot be reclaimed and cannot be used for storing data.

Using the *for load* Option

You can use the **for load** option for recovering from media failure or for moving a database from one machine to another, if you have not added to the database with **sp_addsegment**. Use **alter database for load** to create a new database in the image of the database from which the database dump to be loaded was made. See the *System Administration Guide* for a discussion of duplicating space allocation when loading a dump into a new database.

- When you create a database using the **for load** option, you can run only the following commands in the new database before loading a database dump:
  - **alter database for load**
  - **drop database**
  - **load database**

  After you load the database dump into the new database, you can also use some **dbcc** diagnostic commands in the databases. After you issue the **online database** command, there are no restrictions on the commands you can use.
- A database created with the **for load** option has a status of "don't recover" in the output from **sp_helpdb**.

Getting Information About Databases

- To get a report on a database, execute the system procedure **sp_helpdb**.
- For a report on the space used in a database, use **sp_spaceused**.

Using *with default_location* and *for proxy_update*

Without the **for proxy_update** clause, the behavior of the **with default_location** clause is the same as that provided by the stored procedure **sp_defaultloc** — a default storage location is established for new and existing table creation, but automatic import of proxy table definitions is not done during the processing of the **create database** command.

- If **for proxy_update** is specified with no **default_location**, an error is reported.
- When a proxy database is created (using the **for proxy_update** option), Component Integration Services will be called upon to:
  - Provides an estimate of the database size required to contain all proxy tables representing the actual tables and views found in the primary server's database. This estimate is the number of database pages needed to contain all proxy tables and indexes. The estimate is used if no size is specified, and no database devices are specified.
  - Creates all proxy tables representing the actual tables and views found in the companion server's database.
  - Grants all permissions on proxy tables to *public.*
  - add the *guest* user to the proxy database
  - The database status will be set to indicate that this database 'Is_A_Proxy'. This status is contained in *master.dbo.sysdatabases.status4*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create database permission defaults to System Administrators, who can transfer it to users listed in the *sysusers* table of the master database. However, **create database** permission is often centralized in order to maintain control over database storage allocation.

If you are creating the *sybsecurity* database, you must be a System Security Officer.

create database permission is not included in the **grant all** command.

See Also

Commands	**alter database**, **disk init**, **drop database**, **dump database**, **load database**, **online database**
System procedures	**sp_changedbowner**, **sp_diskdefault**, **sp_helpdb**, **sp_logdevice**, **sp_renamedb**, **sp_spaceused**

create default

Function

Specifies a value to insert in a column (or in all columns of a user-defined datatype) if no value is explicitly supplied at insert time.

Syntax

```
create default [owner.]default_name
   as constant_expression
```

Keywords and Options

default_name – is the name of the default. It must conform to the rules for identifiers and cannot be a variable. Specify the owner's name to create another default of the same name owned by a different user in the current database. The default value for *owner* is the current user.

constant_expression – is an expression that does not include the names of any columns or other database objects. You can include built-in functions that do not reference database objects. Enclose character and date constants in quotes and use a "0x" prefix for binary constants.

Examples

1. `create default phonedflt as "UNKNOWN"`

   Defines a default value. Now, you need to bind it to the appropriate column or user-defined datatype with **sp_bindefault**.

2. `sp_bindefault phonedflt, "authors.phone"`

   The default takes effect only if there is no entry in the *phone* column of the *authors* table. No entry is different from a null value entry. To get the default, issue an **insert** command with a column list that does not include the column that has the default.

3. `create default todays_date as getdate()`

   Creates a default value, *todays_date*, that inserts the current date into the columns to which it is bound.

Comments

- Bind a default to a column or user-defined datatype—but not a Adaptive Server-supplied datatype—with **sp_bindefault**.

- You can bind a new default to a datatype without unbinding the old one. The new default overrides and unbinds the old one.
- To hide the source test of a default, use **sp_hidetext**.

Restrictions

- You can create a default only in the current database.
- **create default** statements cannot be combined with other statements in a single batch.
- You must drop a default with **drop default** before you create a new one of the same name, and you must unbind a default (with the system procedure **sp_unbindefault**) before you drop it.

Datatype Compatibility

- Adaptive Server generates an error message when it tries to insert a default value that is not compatible with the column's datatype. For example, if you bind a character expression such as "N/A" to an *integer* column, any **insert** that does not specify the column value fails.
- If a default value is too long for a character column, Adaptive Server either truncates the string or generates an exception, depending on the setting of the **string_rtruncation** option. For more information, see the **set** command.

Getting Information About Defaults

- Default definitions are stored in *syscomments*.
- After a default is bound to a column, its object ID is stored in *syscolumns*. After a default is bound to a user-defined datatype, its object ID is stored in *systypes*.
- To rename a default, use **sp_rename**.
- For a report on the text of a default, use **sp_helptext**.

Defaults and Rules

- If a column has both a default and a rule associated with it, the default value must not violate the rule. A default that conflicts with a rule cannot be inserted. Adaptive Server generates an error message each time it attempts to insert such a default.

Defaults and Nulls

- If a column does not allow nulls, and you do not create a default for the column, when a user attempts to insert a row but does not include a value for that column, the insert fails and Adaptive Server generates an error message.

  Table 6-8 illustrates the relationship between the existence of a default and the definition of a column as NULL or NOT NULL.

Table 6-8: Relationship between nulls and column defaults

Column Null Type	No Entry, No Default	No Entry, Default Exists	Entry Is Null, No Default	Entry Is Null, Default Exists
NULL	Null inserted	Default value inserted	Null inserted	Null inserted
NOT NULL	Error, command fails	Default value inserted	Error, command fails	Error, command fails

Specifying a Default Value in *create table*

- You can define column defaults using the **default** clause of the **create table** statement as an alternative to using **create default**. However, these column defaults are specific to that table; you cannot bind them to other tables. See **create table** and **alter table** for information about integrity constraints.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Transact-SQL extension	Use the **default** clause of the **create table** statement to create defaults that are SQL92-compliant.

Permissions

create default permission defaults to the Database Owner, who can transfer it to other users.

See Also

Commands	**alter table**, **create rule**, **create table**, **drop default**, **drop rule**
System procedures	**sp_bindefault**, **sp_help**, **sp_helptext**, **sp_rename**, **sp_unbindefault**

create existing table

(Component Integration Services only)

Function

Creates a proxy table, then retrieves and stores metadata from a remote table and places the data into the proxy table. Allows you to map the proxy table to a table, view, or procedure at a remote location.

Syntax

```
create existing table table_name (column_list)
   [ on segment_name ]
   [ [ external {table | procedure} ] at pathname ]
```

Keywords and Options

table_name – specifies the name of the table for which you want to create a proxy table.

column_list – specifies the name of the column list that stores information about the remote table.

on *segment_name* – specifies the segment that contains the remote table.

external – specifies that the object is a remote object.

table – specifies that the remote object is a table or a view. The default is **external table**.

procedure – specifies that the remote object is a stored procedure.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object

where:

- *server_name* (required) is the name of the server that contains the remote object
- *dbname* (optional) is the name of the database managed by the remote server that contains this object
- *owner* (optional) is the name of the remote server user that owns the remote object

- *object* (required) is the name of the remote table, view, or procedure

Examples

1.
```
create existing table authors
(
au_id        id,
au_lname     varchar(40)    NOT NULL,
au_fname     varchar(20)    NOT NULL,
phone        char(12),
address      varchar(40)    NULL,
city         varchar(20)    NULL,
state        char(2)        NULL,
zip          char(5)        NULL,
contract     bit
)
```

   Creates the proxy table *authors*.

2.
```
create existing table syb_columns
(
id          int,
number      smallint,
colid       tinyint,
status      tinyint,
type        tinyint,
length      tinyint,
offset      smallint,
usertype    smallint,
cdefault    int,
domain      int,
name        varchar(30),
printfmt    varchar(255)    NULL,
prec        tinyint         NULL,
scale       tinyint         NULL
)
```

   Creates the proxy table *syb_columns*.

3.
```
create existing table blurbs
(author_id  id      not null,
copy        text    not null)
at "SERVER_A.db1.joe.blurbs"
```

   Creates a proxy table named *blurbs* for the *blurbs* table at the remote server SERVER_A.

```
4. create existing table rpc1
   (column_1   int,
   column_2    int)
   external procedure
   at "SERVER_A.db1.joe.p1"
```

Creates a proxy table named *rpc1* for the remote procedure named *p1*.

Comments

- The **create existing table** command does not create a new table. Instead, Component Integration Services checks the table mapping to confirm that the information in *column_list* matches the remote table, verifies the existence of the underlying object, and retrieves and stores meta data about the remote table.
- If the host data file or remote server object does not exist, the command is rejected with an error message.
- If the object exists, the system tables *sysobjects*, *syscolumns*, and *sysindexes* are updated. The verification is a three-step operation:
  - The nature of the existing object is determined. For host data files, this requires determining file organization and record format. For remote server objects, this requires determining whether the object is a table, a view, or an RPC.
  - For remote server objects (other than RPCs), column attributes obtained for the table or view are compared with those defined in the *column_list*.
  - Index information from the host data file or remote server table is extracted and used to create rows for the system table *sysindexes*. This defines indexes and keys in Adaptive Server terms and enables the query optimizer to consider any indexes that might exist on this table.
- The **on *segment_name*** clause is processed locally and is not passed to a remote server.
- After successfully defining an existing table, issue an **update statistics** command for the table. This allows the query optimizer to make intelligent choices regarding index selection and join order.
- Component Integration Services allows you to create a proxy table with a column defined as NOT NULL even though the remote column is defined as NULL. It displays a warning to notify you of the mismatch.

- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.
- Component Integration Services inserts or updates a record in the *systabstats* catalog for each index of the remote table. Since detailed structural statistics are irrelevant for remote indexes, only a minimum number of columns are set in the *systabstats* record—*id*, *indid*, and *rowcnt*.

Datatype Conversions

- When using the **create existing table** command, you must specify all datatypes with recognized Adaptive Server datatypes. If the remote server tables reside on a class of server that is heterogeneous, the datatypes of the remote table are automatically converted into the specified Adaptive Server types when the data is retrieved. If the conversion cannot be made, Component Integration Services does not allow the table to be defined.
- The *Component Integration Services User's Guide* contains a section for each supported server class and identifies all possible datatype conversions that are implicitly performed by Component Integration Services.

Changes by Server Class

- All server classes now allow you to specify fewer columns than there are in the table on the remote server.
- All server classes now match the columns by name. Some server classes previously matched columns by column ID.
- All server classes now allow the column type to be any datatype that can be converted to and from the datatype of the column in the remote table.

Remote Procedures

- When the proxy table is a procedure-type table, you must provide a column list that matches the description of the remote procedure's result set. **create existing table** does **not** verify the accuracy of this column list.
- No indexes are created for procedures.
- Component Integration Services treats the result set of a remote procedure as a virtual table that can be sorted, joined with other

tables, or inserted into another table using **insert** or **select.** However, a procedure type table is considered read-only, which means you cannot issue the following commands against the table:

- **delete**
- **update**
- **insert**
- **create index**
- **truncate table**
- **alter table**

- Begin the column name with an underscore (_) to specify that the column is not part of the remote procedure's result set. These columns are referred to as parameter columns. For example:

```
create existing table rpc1
(
    a       int,
    b       int,
    c       int,
    _p1     int null,
    _p2     int null
)
external procedure
at "SYBASE.sybsystemprocs.dbo.myproc"
```

  In this example, the parameter columns *_p1* and *_p2* are input parameters. They are not expected in the result set, but can be referenced in the query:

```
select a, b, c from t1
where _p1 = 10 and _p2 = 20
```

  CIS passes the search arguments to the remote procedure as parameters, using the names *@p1* and *@p2*.

- Parameter column definitions in a **create existing table** statement must follow these rules:
  - Parameter column definitions must allow a null value.
  - Parameter columns cannot precede regular result columns—they must appear at the end of the column list.
- If a parameter column is included in a **select** list **and** is passed to the remote procedure as a parameter, the return value is assigned by the **where** clause.

- If a parameter column is included in a **select** list, but does not appear in the **where** clause or cannot be passed to the remote procedure as a parameter, its value is NULL.
- A parameter column can be passed to a remote procedure as a parameter if the Adaptive Server query processor considers it a searchable argument. A parameter column is considered a searchable argument if it is not included in any **or** predicates. For example, the **or** predicate in the second line of the following query prevents the parameter columns from being used as parameters:

```
select a, b, c from t1
where _p1 = 10 or _p2 = 20
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create existing table permission defaults to the table owner and is not transferable.

See Also

Commands	**alter table**, **create table**, **create proxy_table**, **drop index**, **insert**, **order by Clause**, **set**, **update**

create index

Function

Creates an index on one or more columns in a table; creates an index in ascending or descending order for each column; allows up to 31 columns per index; leaves a specified number of unused pages during index creation; allows specification of the number of steps in the distribution histogram for the index.

Syntax

```
create [unique] [clustered | nonclustered]
       index index_name
   on [[database.]owner.]table_name
       (column_name [asc | desc]
       [, column_name [asc | desc]]...)
   [with {
       {fillfactor = pct| max_rows_per_page=num_rows},
       reservepagegap = num_pages,
       consumers = x, ignore_dup_key, sorted_data,
       [ignore_dup_row | allow_dup_row],
       , statistics using num_steps values  } ]
   [on segment_name]
```

Keywords and Options

unique – prohibits duplicate index values (also called “key values”). The system checks for duplicate key values when the index is created (if data already exists), and each time data is added with an **insert** or **update**. If there is a duplicate key value or if more than one row contains a null value, the command fails, and Adaptive Server prints an error message giving the duplicate entry.

◆ ***WARNING!***

Adaptive Server does not detect duplicate rows if a table contains any non-null *text* or *image* columns.

update and **insert** commands that generate duplicate key values fail, unless the index was created with **ignore_dup_row** or **ignore_dup_key.**

Composite indexes (indexes in which the key value is composed of more than one column) can also be unique.

The default is nonunique. To create a nonunique clustered index on a table that contains duplicate rows, you must specify **allow_dup_row** or **ignore_dup_row**. See "Duplicate Rows", below.

clustered – means that the physical order of rows on the current database device is the same as the indexed order of the rows. The bottom, or **leaf level**, of the clustered index contains the actual data pages. A clustered index almost always retrieves data faster than a nonclustered index. Only one clustered index per table is permitted. See "Creating Clustered Indexes", below.

If **clustered** is not specified, **nonclustered** is assumed.

nonclustered – means that the physical order of the rows is not the same as their indexed order. The leaf level of a nonclustered index contains pointers to rows on data pages. You can have up to 249 nonclustered indexes per table.

index_name – is the name of the index. Index names must be unique within a table, but need not be unique within a database.

table_name – is the name of the table in which the indexed column or columns are located. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

column_name – is the column or columns to which the index applies. Composite indexes are based on the combined values of up to 16 columns. The sum of the maximum lengths of all the columns used in a composite index cannot exceed 600 bytes. List the columns to be included in the composite index (in the order in which they should be sorted) inside the parentheses following *table_name*.

asc | desc – specifies whether the index is to be created in ascending or descending order for the column specified. The default is ascending order.

fillfactor – specifies how full Adaptive Server will make each page when it is creating a new index on existing data. The **fillfactor** percentage is relevant only at the time the index is created. As the data changes, the pages are not maintained at any particular level of fullness.

The default for **fillfactor** is 0; this is used when you do not include **with fillfactor** in the **create index** statement (unless the value has been changed with **sp_configure**). When specifying a **fillfactor**, use a value between 1 and 100.

A **fillfactor** of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes. There is seldom a reason to change the **fillfactor**.

If the **fillfactor** is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A **fillfactor** of 100 only makes sense for read-only tables—tables to which no additional data will ever be added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A **fillfactor** of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small **fillfactor** values cause each index (or index and data) to take more storage space.

◆ ***WARNING!***

Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

max_rows_per_page – limits the number of rows on data pages and the leaf level pages of indexes. **max_rows_per_page** and **fillfactor** are mutually exclusive. Unlike **fillfactor**, the **max_rows_per_page** value is maintained until it is changed with **sp_chgattribute**.

If you do not specify a value for **max_rows_per_page**, Adaptive Server uses a value of 0 when creating the table. Values for tables and clustered indexes are between 0 and 256. The maximum number of rows per page for nonclustered indexes depends on the size of the index key. Adaptive Server returns an error message if the specified value is too high.

A **max_rows_per_page** value of 0 creates clustered indexes with full pages and nonclustered indexes with full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes.

If **max_rows_per_page** is set to 1, Adaptive Server creates both clustered and nonclustered indexes with one row per page at the leaf level. Use low values to reduce lock contention on frequently accessed data. However, low **max_rows_per_page** values cause Adaptive Server to create new indexes with pages that are not completely full, uses more storage space, and may cause more page splits.

If Component Integration Services is enabled, you cannot use **max_rows_per_page** for remote servers.

◆ ***WARNING!***

Creating a clustered index with max_rows_per_page can affect the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

with reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the index. Valid values are 0–255. The default is 0.

ignore_dup_key – cancels attempts of duplicate key entry into a table that has a unique index (clustered or nonclustered). Adaptive Server cancels the attempted **insert** or **update** of a duplicate key with an informational message. After the cancellation, the transaction containing the duplicate keys proceeds to completion.

You cannot create a unique index on a column that includes duplicate values or more than one null value, whether or not **ignore_dup_key** is set. If you attempt to do so, Adaptive Server prints an error message that gives the first of the duplicate values. You must eliminate duplicates before Adaptive Server can create a unique index on the column.

ignore_dup_row – allows you to create a new, nonunique clustered index on a table that includes duplicate rows by deleting the duplicate rows from the table, and cancels any **insert** or **update** that would create a duplicate row, but does not roll back the entire transaction. See “Duplicate Rows”, below, for more information.

allow_dup_row – allows you to create a nonunique clustered index on a table that includes duplicate rows, and allows you to duplicate rows with **update** and **insert** statements. See “Duplicate Rows”, below, for an explanation of how to use these options.

sorted_data – speeds creation of clustered indexes or unique nonclustered indexes when the data in the table is already in sorted order (for example, when you have used **bcp** to copy data that has already been sorted into an empty table). See "Using the sorted_data Option to Speed Sorts" for more information.

with statistics using *num_steps* **values** – specifies the number of steps to generate for the histogram used to optimize queries. If this clause is omitted:

- The default value is 20, if no histogram is currently stored for the leading index column,
- The current number of steps is used, if a histogram for the leading column of the index column already exists.

If you specify 0 for *num_steps*, the index is re-created, but the statistics for the index are not overwritten in the system tables.

on *segment_name* – creates the index on the named segment. Before using the **on** *segment_name* option, initialize the device with **disk init**, and add the segment to the database with the **sp_addsegment** system procedure. See your System Administrator, or use **sp_helpsegment** for a list of the segment names available in your database.

with consumers – specifies the number of consumer processes that should perform the sort operation for creating the index. The actual number of consumer processes used to sort the index may be smaller than the specified number, if fewer worker processes are available when Adaptive Server executes the sort.

Examples

1.
```
create index au_id_ind
on authors (au_id)
```

   Creates an index named *au_id_ind* on the *au_id* column of the *authors* table.

2.
```
create unique clustered index au_id_ind
on authors(au_id)
```

   Creates a unique clustered index named *au_id_ind* on the *au_id* column of the *authors* table.

3.
```
create index ind1
on titleauthor (au_id, title_id)
```

   Creates an index named *ind1* on the *au_id* and *title_id* columns of the *titleauthor* table.

4.
```
create nonclustered index zip_ind
on authors(postalcode)
with fillfactor = 25, consumers = 4
```

   Creates a nonclusters index named *zip_ind* on the *zip* column of the *authors* table, filling each index page one-quarter full and limiting the sort to 4 consumer processes.

5.
```
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
```

   Creates an index with ascending ordering on *pub_id* and descending order on *pubdate*.

6.
```
create index title_id_ix
    on titles (title_id)
    with reservepagegap = 40,
    statistics using 50 values
```

   Creates an index on *title_id*, using 50 histogram steps for optimizer statistics and leaving 1 empty page out of every 40 pages in the index.

Comments

- Run **update statistics** periodically if you add data to the table that changes the distribution of keys in the index. The query optimizer uses the information created by **update statistics** to select the best plan for running queries on the table.
- If the table contains data when you create a nonclustered index, Adaptive Server runs **update statistics** on the new index. If the table contains data when you create a clustered index, Adaptive Server runs **update statistics** on all the table's indexes.
- Index all columns that are regularly used in joins.
- When Component Integration Services is enabled, the **create index** command is reconstructed and passed directly to the Adaptive Server associated with the table.

Restrictions

- You cannot create an index on a column with a datatype of *bit*, *text*, or *image*.
- A table can have a maximum of 249 nonclustered indexes.
- A table can have a maximum of one clustered index.
- You can specify up to 31 columns (formerly 16) for the index key. The maximum total number of bytes is 600.

- You can create an index on a temporary table. It disappears when the table disappears.
- You can create an index on a table in another database, as long as you are the owner of that table.
- You cannot create an index on a view.
- **create index** runs more slowly while a **dump database** is taking place.
- You can create a clustered index on a partitioned table or partition a table with a clustered index if the following conditions are true:
  - The **select into/bulkcopy/pllsort** database option is turned on,
  - Adaptive Server is configured for parallel processing, and
  - There is one more worker process available than the number of partitions.

  For more information about clustered indexes on partitioned tables, see Chapter 13, "Parallel Sorting," in the *Performance and Tuning Guide*.
- The maximum number of indexes allowed on a data-only-locked table with a clustered index is 249. A table can have 1 clustered index and 248 nonclustered indexes.

Creating Indexes Efficiently

- Indexes speed data retrieval, but can slow data updates. For better performance, create a table on one segment and create its nonclustered indexes on another segment, when the segments are on separate physical devices.
- Adaptive Server can create indexes in parallel if a table is partitioned and the server is configured for parallelism. It can also use sort buffers to reduce the amount of I/O required during sorting. For more information, see Chapter 13, "Parallel Sorting," in the *Performance and Tuning Guide*.
- Create a clustered index before creating any nonclustered indexes, since nonclustered indexes are automatically rebuilt when a clustered index is created.
- When using parallel sort for data-only-locked tables, the number of worker processes must be configured to equal or exceed the number of partitions, even for empty tables. The database option **select into/bulkcopy/pllsort** must also be enabled.

Creating Clustered Indexes

- A table "follows" its clustered index. When you create a table, then use the **on** *segment_name* extension to **create clustered index**, the table migrates to the segment where the index is created.

  If you create a table on a specific segment, then create a clustered index without specifying a segment, Adaptive Server moves the table to the default segment when it creates the clustered index there.

  Because text and image data is stored in a separate page chain, creating a clustered index with **on** *segment_name* does not move text and image columns.
- To create a clustered index, Adaptive Server duplicates the existing data; the server deletes the original data when the index is complete. Before creating a clustered index, use **sp_spaceused** to make sure that the database has at least 120 percent of the size of the table available as free space.
- The clustered index is often created on the table's primary key (the column or columns that uniquely identify the row). The primary key can be recorded in the database (for use by front-end programs and the system procedure **sp_depends**) with the system procedure **sp_primarykey.**
- To allow duplicate rows in a clustered index, specify **allow_dup_row.**

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sorting order for the index keys. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing. For more information, see Chapter 9, "Indexing for Performance," in the *Performance and Tuning Guide*.

Space Requirements for Indexes

- Space is allocated to tables and indexes in increments of one extent, or eight pages, at a time. Each time an extent is filled, another extent is allocated. (Use the system procedure **sp_spaceused** to display the amount of space allocated and used by an index.)
- In some cases, using the **sorted_data** option allows Adaptive Server to skip copying the data rows as described in Table 6-11. In these cases, you need only enough additional space for the index

structure itself. Depending on key size, this is usually about 20 percent of the size of the table.

Duplicate Rows

- The **ignore_dup_row** and **allow_dup_row** options are not relevant when you are creating a nonunique, nonclustered index. Because Adaptive Server attaches a unique row identification number internally in each nonclustered index, it never worries about duplicate rows, not even for identical data values.
- **ignore_dup_row** and **allow_dup_row** are mutually exclusive.
- A nonunique clustered index allows duplicate keys, but does not allow duplicate rows unless you specify **allow_dup_row.**
- **allow_dup_row** allows you to create a nonunique, clustered index on a table that includes duplicate rows. If a table has a nonunique, clustered index that was created without the **allow_dup_row** option, you cannot create new duplicate rows using the **insert** or **update** command.

  If any index in the table is unique, the requirement for uniqueness takes precedence over the **allow_dup_row** option. You cannot create an index with **allow_dup_row** if a unique index exists on any column in the table.
- The **ignore_dup_row** option is also used with a nonunique, clustered index. The **ignore_dup_row** option eliminates duplicates from a batch of data. **ignore_dup_row** cancels any **insert** or **update** that would create a duplicate row, but does not roll back the entire transaction.
- Table 6-9 illustrates how **allow_dup_row** and **ignore_dup_row** affect attempts to create a nonunique, clustered index on a table that includes duplicate rows and attempts to enter duplicate rows into a table.

Table 6-9: Duplicate row options for nonunique clustered indexes

Option Setting	Create an Index on a Table That Has Duplicate Rows	Insert Duplicate Rows into a Table With an Index
Neither option set	**create index** fails.	**insert** fails.
allow_dup_row set	**create index** completes.	**insert** completes.
ignore_dup_row set	Index is created but duplicate rows are deleted; error message.	All rows are inserted except duplicates; error message.

Table 6-10 shows which index options can be used with the different types of indexes:

Table 6-10: Index options

Index Type	Options
Clustered	**ignore_dup_row** | **allow_dup_row**
Unique, clustered	**ignore_dup_key**
Nonclustered	None
Unique, nonclustered	**ignore_dup_key**, **ignore_dup_row**

Using Unique Constraints in Place of Indexes

- As an alternative to **create index**, you can implicitly create unique indexes by specifying a unique constraint with the **create table** or **alter table** statement. The unique constraint creates a clustered or nonclustered unique index on the columns of a table. These **implicit** indexes are named after the constraint, and they follow the same rules for indexes created with **create index**.
- You cannot drop indexes supporting unique constraints using the **drop index** statement. They are dropped when the constraints are dropped through an **alter table** statement or when the table is dropped. See **create table** for more information about unique constraints.

Using the *sorted_data* Option to Speed Sorts

- The **sorted_data** option can reduce the time needed to create an index by skipping the sort step and by eliminating the need to copy the data rows to new pages in certain cases. The speed increase becomes significant on large tables and increases to several times faster in tables larger than 1GB.

  If **sorted_data** is specified, but data is not in sorted order, Adaptive Server displays an error message, and the command fails.

  Creating a nonunique, nonclustered index succeeds, unless there are rows with duplicate keys. If there are rows with duplicate keys, Adaptive Server displays an error message, and the command fails.
- The effects of **sorted_data** for creating a clustered index depend on whether the table is partitioned and whether certain other options are used in the **create index** command. Some options require data copying, if used at all, for nonpartitioned tables and sorts plus data copying for partitioned tables, while others require data copying only if you use one of the following options:

  - Using the **ignore_dup_row** option
  - Using the **fillfactor** option
  - Using the **on** *segmentname* clause to specify a segment that is different from the segment where the table data is located
  - Using the **max_rows_per_page** clause to specify a value that is different from the value associated with the table
- Table 6-11 shows when the sort is required and when the table is copied for partitioned and nonpartitioned tables.

Table 6-11: Using the sorted_data option for creating a clustered index

Options	Partitioned Table	Unpartitioned table
No options specified	Parallel sort; copies data, distributing evenly on partitions; creates index tree.	Either parallel or nonparallel sort; copies data, creates index tree.
with sorted_data only or **with sorted_data on** *same_segment*	Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.	Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.
with sorted_data and **ignore_dup_row** or **fillfactor** or **on** *other_segment* or **max_rows_per_page**	Parallel sort; copies data, distributing evenly on partitions; creates index tree.	Copies data and creates the index tree. Does not perform the sort. Does not run in parallel.

Specifying the Number of Histogram Steps

- Use the **with statistics** clause to specify the number of steps for a histogram for the leading column of an index. Histograms are used during query optimization to determine the number of rows that match search arguments for a column.
- To re-create an index without updating the values in *sysstatistics* for a column, use 0 for the number of steps. This avoids overwriting statistics that have been changed with **optdiag**.

Space Management Properties

- **fillfactor**, **max_rows_per_page**, and **reservepagegap** help manage space on index pages in different ways:
  - **fillfactor** applies to indexes for all locking schemes. For clustered indexes on allpages-locked tables, it affects the data pages of

the table. On all other indexes, it affects the leaf level of the index.

- **max_rows_per_page** applies only to index pages of allpages-locked tables.
- **reservepagegap** applies to tables and indexes for all locking schemes.

- **reservepagegap** affects space usage in indexes:
  - At the time the index is created
  - When **reorg** commands on indexes are executed
  - When nonclustered indexes are rebuilt after creating a clustered index
- When a **reservepagegap** value is specified in a **create clustered index** command, it applies:
  - To the data and index pages of allpages-locked tables
  - To only the index pages of data-only-locked tables
- The *num_pages* value specifies a ratio of filled pages to empty pages on the leaf level of the index so that indexes can allocate space close to existing pages, as new space is required. For example, a **reservepagegap** of 10 leaves 1 empty page for each 9 used pages.
- **reservepagegap** specified along with **create clustered index** on an allpages-locked table overwrites any value previously specified with **create table** or **alter table**.
- You can change the space management properties for an index with **sp_chgattribute**. Changing properties with **sp_chgattribute** does not immediately affect storage for indexes on the table. Future large scale allocations, such as running **reorg rebuild**, use the **sp_chgattribute** value.
- The **fillfactor** value set by **sp_chgattribute** is stored in the *fill_factor* column in *sysindexes*. The **fillfactor** is applied when an index is recreated as a result of an **alter table...lock** command or a **reorg rebuild** command.

Index Options and Locking Modes

- Table 6-12 shows the index options supported for allpages-locked and data-only-locked tables. On data-only-locked tables, the **ignore_dup_row** and **allow_dup_row** options are enforced during **create index**, but are not enforced during **insert** and **update** operations.

Data-only-locked tables always allow the insertion of duplicate rows.

Table 6-12: create index options supported for locking schemes

Index Type	Allpages-Locked Table	Data-Only-Locked Table	
		During Index Creation	During Inserts
Clustered	**allow_dup_row** **ignore_dup_row**	**allow_dup_row** **ignore_dup_row**	**allow_dup_row**
Unique clustered	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**
Nonclustered	None	None	None
Unique nonclustered	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**

Table 6-13 shows the behavior of commands that attempt to insert duplicate rows into tables with clustered indexes, and when the clustered indexes are dropped and re-created.

Table 6-13: Enforcement and errors for duplicate row options

Options	Allpages-Locked Table	Data-Only-Locked Table
No options specified	Insert fails with error message 2615. Re-creating the index succeeds.	Insert succeeds. Re-creating the index fails with error message 1508.
allow_dup_row	Insert and re-creating the index succeed.	Insert and re-creating the index succeed.
ignore_dup_row	Insert fails with "Duplicate row was ignored" message. Re-creating the index succeeds.	Insert succeeds. Re-creating the index deletes duplicate rows.

Using the *sorted_data* Option on Data-Only-Locked Tables

- The **sorted_data** option to **create index** can be used only immediately following a bulk copy operation into an empty table. Once data modifications to that table cause additional page allocations, the **sorted_data** option cannot be used.
- Specifying different values for space management properties may override the sort suppression functionality of the **sorted_data**.

Getting Information About Tables and Indexes

- Each index—including composite indexes—is represented by one row in *sysindexes*.
- For information about the order of the data retrieved through indexes and the effects of an Adaptive Server's installed sort order, see the **order by** clause.
- For information about a table's indexes, execute the system procedure **sp_helpindex**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create index permission defaults to the table owner and is not transferable.

See Also

Commands	**alter table**, **create table**, **drop index**, **insert**, **order by Clause**, **set**, **update**
System procedures	**sp_addsegment**, **sp_chgattribute**, **sp_helpindex**, **sp_helpsegment**, **sp_spaceused**
Utility Program	**optdiag**

create plan

Function

Creates an abstract plan.

Syntax

```
create plan query plan
       [into group_name]
       [and set @new_id}
```

Keywords and Options

query – is a string literal, parameter, or local variable containing the SQL text of a query.

plan – is a string literal, parameter, or local variable containing an abstract plan expression.

into *group_name* – specifies the name of an abstract plan group.

and set @*new_id* – returns the ID number of the abstract plan in the variable.

Examples

1.
```
create plan "select * from titles where price >
$20" "(t_scan titles)"
```

   Creates an abstract plan for the specified query.

2.
```
declare @id int
create plan "select au_fname, au_lname from
authors where au_id = '724-08-9931' "
"(i_scan au_id_ix authors)"
into dev_plans
and set @id
select @id
```

   Creates an abstract plan for the query in the *dev_plans* group, and returns the plan ID in the variable *@id*.

Comments

- **create plan** saves the abstract plan in the group specified with **into**. If no group name is specified, it saves the plan in the currently active plan group.

- Queries and abstract plans specified with **create plan** are not checked for valid SQL syntax and plans are not checked for valid abstract plan syntax. Also, the plan is not checked for compatibility with the SQL text. All plans created with **create plan** should be immediately checked for correctness by running the query specified in the **create plan** statement.
- If another query plan in the group has the same SQL text, the **replace** mode must be enabled with **set plan replace on**. Otherwise, the **create plan** command fails.
- You must declare *@new_id* before using it in the **and set** clause.
- The abstract plan group you specify with **into** must already exist.

Standards and Compliance

create plan is a Transact-SQL extension.

Permissions

create plan permission defaults to all users. No permission is required to use it.

See Also

Commands	**set plan**
System procedures	**sp_add_qpgroup**, **sp_find_qplan**, **sp_help_qpgroup**, **sp_set_qplan**

create procedure

Function

Creates a stored procedure or an extended stored procedure (ESP) that can take one or more user-supplied parameters.

Syntax

```
create procedure [owner.]procedure_name[;number]
   [[(]@parameter_name
       datatype [(length) | (precision [, scale])]
       [= default][output]
   [, @parameter_name
       datatype [(length) | (precision [, scale])]
       [= default][output]]...[)]]
   [with recompile]
   as {SQL_statements | external name dll_name}
```

Keywords and Options

procedure_name – is the name of the procedure. It must conform to the rules for identifiers and cannot be a variable. Specify the owner's name to create another procedure of the same name owned by a different user in the current database. The default value for *owner* is the current user.

;number – is an optional integer used to group procedures of the same name so that they can be dropped together with a single **drop procedure** statement. Procedures used in the same application are often grouped this way. For example, if the procedures used with the application named **orders** are named *orderproc;1*, *orderproc;2*, and so on, the statement:

```
drop proc orderproc
```

drops the entire group.

Once procedures have been grouped, individual procedures within the group cannot be dropped. For example, the statement:

```
drop procedure orderproc;2
```

is not allowed.

You cannot group procedures if you are running Adaptive Server in the **evaluated configuration**. The evaluated configuration requires that you disallow procedure grouping so that every stored procedure has a unique object identifier and

can be dropped individually. To disallow procedure grouping, a System Security Officer must reset the configuration parameter **allow procedure grouping** with the system procedure **sp_configure**. For more information about the evaluated configuration, see the *System Administration Guide*.

parameter_name – is the name of an argument to the procedure. The value of each parameter is supplied when the procedure is executed. (Parameter names are optional in **create procedure** statements—a procedure need not take any arguments.)

Parameter names must be preceded by the @ sign and conform to the rules for identifiers. A parameter name, including the @ sign, can be a maximum of 30 characters. Parameters are local to the procedure: the same parameter names can be used in other procedures.

If the value of a parameter contains non-alphanumeric characters, it must be enclosed in quotes. This includes object names qualified by a database name or owner name, since they include a period. If the value of a character parameter begins with a numeric character, it also must be enclosed in quotes.

datatype [(*length*) | (*precision* [, *scale*])] – is the datatype of the parameter. See Chapter 1, "System and User-Defined Datatypes," for more information about datatypes. Stored procedure parameters cannot have a datatype of *text* or *image* or a user-defined datatype whose underlying type is *text* or *image*.

The *char*, *varchar*, *nchar*, *nvarchar*, *binary*, and *varbinary* datatypes should include a *length* in parentheses. If you omit the length, Adaptive Server truncates the parameter value to 1 character.

The *float* datatype expects a binary *precision* in parentheses. If you omit the precision, Adaptive Server uses the default precision for your platform.

The *numeric* and *decimal* datatypes expect a *precision* and *scale*, enclosed in parentheses and separated by a comma. If you omit the precision and scale, Adaptive Server uses a default precision of 18 and a scale of 0.

default – defines a default value for the procedure's parameter. If a default is defined, a user can execute the procedure without giving a parameter value. The default must be a constant. It can include the wildcard characters (%, _, [ ], and [^]) if the procedure uses the parameter name with the keyword **like** (see example 2).

The default can be NULL. The procedure definition can specify that some action be taken if the parameter value is NULL (see example 3).

output – indicates that the parameter is a return parameter. Its value can be returned to the **execute** command that called this procedure. Use return parameters to return information to the calling procedure (see example 5).

To return a parameter value through several levels of nested procedures, each procedure must include the **output** option with the parameter name, including the **execute** command that calls the highest level procedure.

The **output** keyword can be abbreviated to **out**.

with recompile – means that Adaptive Server never saves a plan for this procedure; a new plan is created each time it is executed. Use this optional clause when you expect that the execution of a procedure will be atypical—that is, when you need a new plan. The **with recompile** clause has no impact on the execution of an extended stored procedure.

SQL_statements – specify the actions the procedure is to take. Any number and kind of SQL statements can be included, with the exception of **create view**, **create default**, **create rule**, **create procedure**, **create trigger**, and **use**.

create procedure SQL statements often include control-of-flow language, including one or more of the following: **declare**; **if...else**; **while**; **break**; **continue**; **begin...end**; **goto** label; **return**; **waitfor**; **/*** *comment* ***/**. They can also refer to parameters defined for the procedure.

The SQL statements can reference objects in another database, as long as they are properly qualified.

- **external name** – creates an extended stored procedure. If the **as external name** syntax is used, you cannot use the *number* parameter with **as external name**.

dll_name – specifies the name of the dynamic link library (DLL) or shared library containing the functions that implement the extended stored procedure. The *dll_name* can be specified with no extension or with a platform-specific extension, such as *.dll* on Windows NT or *.so* on Sun Solaris. If you specify the extension, enclose the entire *dll_name* in quotation marks.

Examples

1.
```
create procedure showind @tabname varchar(30)
as
  select sysobjects.name, sysindexes.name, indid
  from sysindexes, sysobjects
  where sysobjects.name = @tabname
  and sysobjects.id = sysindexes.id
```

   Given a table name, the procedure *showind* displays its name and the names and identification numbers of any indexes on any of its columns.

   Here are the acceptable syntax forms for executing *showind*:

```
execute showind titles
```

```
execute showind @tabname = "titles"
```

   Or, if this is the first statement in a file or batch:

```
showind titles
```

2.
```
create procedure
showsysind @table varchar(30) = "sys%"
as
  select sysobjects.name, sysindexes.name, indid
  from sysindexes, sysobjects
  where sysobjects.name like @table
  and sysobjects.id = sysindexes.id
```

   This procedure displays information about the system tables if the user does not supply a parameter.

3.
```
create procedure
showindnew @table varchar(30) = null
as
  if @table is null
   print "Please give a table name"
  else
   select sysobjects.name, sysindexes.name, indid
   from sysindexes, sysobjects
   where sysobjects.name = @table
   and sysobjects.id = sysindexes.id
```

   This procedure specifies an action to be taken if the parameter is NULL (that is, if the user does not give a parameter).

4.
```
create procedure mathtutor @mult1 int, @mult2 int,
  @result int output
as
select @result = @mult1 * @mult2
```

   This procedure multiplies two integer parameters and returns the product in the **output** parameter, *@result*. If the procedure is executed by passing it 3 integers, the **select** statement performs the multiplication and assigns the values, but does not print the return parameter:

```
mathtutor 5, 6, 32

(return status 0)
```

5.
```
declare @guess int
select @guess = 32
exec mathtutor 5, 6, @result = @guess output

(1 row affected)
(return status = 0)

Return parameters:

@result
-----------
         30
```

   In this example, both the procedure and the **execute** statement include **output** with a parameter name so that the procedure can return a value to the caller. The output parameter and any subsequent parameters in the **execute** statement, *@result*, **must** be passed as:

```
@parameter = value
```

   - The value of the return parameter is always reported, whether or not its value has changed.
   - *@result* does not need to be declared in the calling batch because it is the name of a parameter to be passed to *mathtutor*.
   - Although the changed value of *@result* is returned to the caller in the variable assigned in the **execute** statement (in this case, *@guess*), it is displayed under its own heading (*@result*).

```
6. declare @guess int
   declare @store int
   select @guess = 32
   select @store = @guess
   execute mathtutor 5, 6, @result = @guess output
   select Your_answer = @store, Right_answer = @guess
   if @guess = @store
       print "Right-o"
   else
       print "Wrong, wrong, wrong!"

   (1 row affected)
   (1 row affected)
   (return status = 0)

   Return parameters:

   @result
   -----------
            30

   Your_answer Right_answer
   ----------- ------------
            32           30

   (1 row affected)
   Wrong, wrong, wrong!
```

Return parameters can be used in additional SQL statements in the batch or calling procedure. This example shows how to use the value of *@guess* in conditional clauses after the **execute** statement by storing it in another variable name, *@store*, during the procedure call. When return parameters are used in an **execute** statement that is part of a SQL batch, the return values are printed with a heading before subsequent statements in the batch are executed.

```
7. create procedure xp_echo @in varchar(255),
       @out varchar(255) output
   as external name "sqlsrvdll.dll"
```

Creates an extended stored procedure named *xp_echo*, which takes an input parameter, *@in*, and echoes it to an output parameter, *@out*. The code for the procedure is in a function named **xp_echo**, which is compiled and linked into a DLL named *sqlsrvdll.dll*.

Comments

- After a procedure is created, you can run it by issuing the **execute** command along with the procedure's name and any parameters. If a procedure is the first statement in a batch, you can give its name without the keyword **execute**.
- You can hide the source text for a procedure, which is stored in *syscomments*, with **sp_hidetext**.
- When a stored procedure batch executes successfully, Adaptive Server sets the *@@error* global variable to 0.

Restrictions

- The maximum number of parameters that a stored procedure can have is 255.
- The maximum number of local and global variables in a procedure is limited only by available memory.
- The maximum amount of text in a stored procedure is 16MB.
- A **create procedure** statement cannot be combined with other statements in a single batch.
- You can create a stored procedure only in the current database, although the procedure can reference objects from other databases. Any objects referenced in a procedure must exist at the time you create the procedure. You can create an object within a procedure, then reference it, provided the object is created before it is referenced.

  You cannot use **alter table** in a procedure to add a column and then refer to that column within the procedure.
- If you use **select** * in your **create procedure** statement, the procedure (even if you use the **with recompile** option to **execute**) does not pick up any new columns you may have added to the table. You must **drop** the procedure and re-create it.
- Within a stored procedure, you cannot create an object (including a temporary table), drop it, then create a new object with the same name. Adaptive Server creates the objects defined in a stored procedure when the procedure is executed, not when it is compiled.

◆ ***WARNING!***

Certain changes to databases, such as dropping and re-creating indexes, can cause object IDs to change. When object IDs change, stored procedures recompile automatically, and can increase slightly in size. Leave some space for this increase.

Extended Stored Procedures

- If the **as** *external name* syntax is used, **create procedure** registers an extended stored procedure (ESP). Extended stored procedures execute procedural language functions rather than Transact-SQL commands.
- On Windows NT, an ESP function should not call a C run-time signal routine. This can cause XP Server to fail, because Open Server™ does not support signal handling on Windows NT.
- To support multi-threading, ESP functions should use the Open Server **srv_yield** function, which suspends and reschedules the XP Server thread to allow another thread of the same or higher priority to execute.
- The DLL search mechanism is platform-dependent. On Windows NT, the sequence of a DLL file name search is as follows:

  a. The directory from which the application is loaded

  b. The current directory

  c. The system directory (SYSTEM32)

  d. Directories listed in the PATH environment variable

  If the DLL is not in the first three directories, set the PATH to include the directory in which it is located.

  On UNIX platforms, the search method varies with the particular platform. If it fails to find the DLL or shared library, it searches *$SYBASE/lib*.

  Absolute path names are not supported.

System Procedures

- System Administrators can create new system procedures in the *sybsystemprocs* database. System procedure names must begin with the characters “sp_”. These procedures can be executed from any database by specifying the procedure name; it is not

necessary to qualify it with the *sybsystemprocs* database name. For more information about creating system procedures, see the *System Administration Guide*.

- System procedure results may vary depending on the context in which they are executed. For example, the system procedure *sp_foo*, which executes the **db_name()** system function, returns the name of the database from which it is executed. When executed from the *pubs2* database, it returns the value "pubs2":

```
use pubs2
sp_foo
------------------------------
pubs2
```

When executed from *sybsystemprocs*, it returns the value "sybsystemprocs":

```
use sybsystemprocs
sp_foo
------------------------------
sybsystemprocs
```

Procedure Return Status

- Stored procedures can return an integer value called a **return status**. The return status either indicates that the procedure executed successfully or specifies the type of error that occurred.
- When you execute a stored procedure, it automatically returns the appropriate status code. Adaptive Server currently returns the following status codes:

Code	Meaning
0	Procedure executed without error
-1	Missing object
-2	Datatype error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Non-fatal internal problem
-9	System limit was reached
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt

Code	Meaning
-14	Hardware error

Codes -15 through -99 are reserved for future use.

- Users can generate a user-defined return status with the **return** statement. The status can be any integer other than 0 through -99. The following example returns "1" when a book has a valid contract and "2" in all other cases:

```
create proc checkcontract @titleid tid
as
if (select contract from titles where
        title_id = @titleid) = 1
   return 1
else
   return 2

checkcontract @titleid = "BU1111"

(return status = 1)

checkcontract @titleid = "MC3026"

(return status = 2)
```

- If more than one error occurs during execution, the code with the highest absolute value is returned. User-defined return values take precedence over system-defined values.

Object Identifiers

- To change the name of a stored procedure, use **sp_rename**.
- To change the name of an extended stored procedure, drop the procedure, rename and recompile the supporting function, then recreate the procedure.
- If a procedure references table names, column names, or view names that are not valid identifiers, you must **set quoted_identifier on** before the **create procedure** command and enclose each such name in double quotes. The **quoted_identifier** option does **not** need to be on when you execute the procedure.
- You must drop and re-create the procedure if any of the objects it references have been renamed.
- Inside a stored procedure, object names used with the **create table** and **dbcc** commands must be qualified with the object owner's name if other users are to make use of the stored procedure. For example, user "mary," who owns table *marytab*, should qualify the name of her table inside a stored procedure (when it is used

with these commands) if she wants other users to be able to execute it. This is because the object names are resolved when the procedure is run. When another user tries to execute the procedure, Adaptive Server looks for a table called *marytab* owned by the user "mary" and not a table called *marytab* owned by the user executing the stored procedure.

Object names used with other statements (for example, **select** or **insert**) inside a stored procedure need not be qualified because the names are resolved when the procedure is compiled.

Temporary Tables and Procedures

- You can create a procedure to reference a temporary table if the temporary table is created in the current session. A temporary table created within a procedure disappears when the procedure exits. For more information, see the *Transact-SQL User's Guide*.
- System procedures such as **sp_help** work on temporary tables, but only if you use them from *tempdb*.

Setting Options in Procedures

- You can use the **set** command inside a stored procedure. Most **set** options remain in effect during the execution of the procedure, then revert to their former settings.

  However, if you use a **set** option (such as **identity_insert**) which requires the user to be the object owner, a user who is not the object owner cannot execute the stored procedure.

Getting Information About Procedures

- For a report on the objects referenced by a procedure, use **sp_depends**.
- To display the text of a **create procedure** statement, which is stored in *syscomments*, use the system procedure **sp_helptext** with the procedure name as the parameter. You must be using the database where the procedure resides when you use **sp_helptext**. To display the text of a system procedure, execute **sp_helptext** from the *sybsystemprocs* database.
- To see a list of system extended stored procedures and their supporting DLLs, use **sp_helpextendedproc** from the *sybsystemprocs* database.

Nested Procedures

- Procedure nesting occurs when one stored procedure calls another.
- If you execute a procedure that calls another procedure, the called procedure can access objects created by the calling procedure.
- The nesting level increments when the called procedure begins execution and decrements when the called procedure completes execution. Exceeding the maximum of 16 levels of nesting causes the transaction to fail.
- You can call another procedure by name or by a variable name in place of the actual procedure name.
- The current nesting level is stored in the *@@nestlevel* global variable.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create procedure permission defaults to the Database Owner, who can transfer it to other users.

Permission to use a procedure must be granted explicitly with the **grant** command and may be revoked with the **revoke** command.

Permissions on Objects: Procedure Creation Time

When you create a procedure, Adaptive Server makes no permission checks on objects, such as tables and views, that are referenced by the procedure. Therefore, you can create a procedure successfully even though you do not have access to its objects. All permission checks occur when a user executes the procedure.

Permissions on Objects: Procedure Execution Time

When the procedure is executed, permission checks on objects depend upon whether the procedure and all referenced objects are owned by the same user.

- If the procedure's objects are owned by different users, the invoker must have been granted direct access to the objects. For

example, if the procedure performs a select from a table that the user cannot access, the procedure execution fails.

- If a procedure and its objects are owned by the same user, however, special rules apply. The invoker automatically has "implicit permission" to access the procedure's objects even though the invoker could not access them directly. Without having to grant users direct access to your tables and views, you can give them restricted access with a stored procedure. In this way, a stored procedure can be a security mechanism. For example, invokers of the procedure might be able to access only certain rows and columns of your table.

  A detailed description of the rules for implicit permissions is discussed in the *System Administration Guide*.

See Also

Commands	**begin...end**, **break**, **continue**, **declare**, **drop procedure**, **execute**, **goto Label**, **grant**, **if...else**, **return**, **select**, **waitfor**, **while**
System procedures	**sp_addextendedproc**, **sp_helpextendedproc**, **sp_helptext**, **sp_hidetext**, **sp_rename**

create proxy_table

(Component Integration Services only)

Function

Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

Syntax

```
create proxy_table table_name
   [ external table ]
   at pathname
```

Keywords and Options

table_name – specifies the local proxy table name to be used by subsequent statements. *table_name* takes the form:

dbname.owner.object

where *dbname* and *owner* are optional and represent the local database and owner name. If *dbname* is not specified, the table is created in the current database; if *owner* is not specified, the table is owned by the current user. If either *dbname* or *owner* is specified, the entire *table_name* must be enclosed in quotes. If only *dbname* is present, a placeholder is required for *owner*.

external table – specifies that the object is a remote table or view. **external table** is the default, so this clause is optional.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object

where:

- *server_name* (required) is the name of the server that contains the remote object
- *dbname* (optional) is the name of the database managed by the remote server that contains this object
- *owner* (optional) is the name of the remote server user that owns the remote object
- *object* (required) is the name of the remote table or view

Examples

1.
```
create proxy_table t1
at "SERVER_A.db1.joe.t1"
```

   Creates a proxy table named *t1* that is mapped to the remote table *t1*. CIS derives the column list from the remote table.

Comments

- **create proxy_table** is a variant of the **create existing table** command. You use **create proxy_table** to create a proxy table, but (unlike **create existing table**) you do not specify a column list. CIS derives the column list from the metadata it obtains from the remote table.
- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.
- If the remote server object does not exist, the command is rejected with an error message.
- If the object exists, the local system tables are updated. Every column is used. Columns and their attributes are obtained for the table or view.
- CIS automatically converts the datatype of the column into an Adaptive Server datatype. If the conversion cannot be made, the **create proxy_table** command does not allow the table to be defined.
- Index information from the remote server table is extracted and used to create rows for the system table *sysindexes*. This defines indexes and keys in Adaptive Server terms and enables the query optimizer to consider any indexes that may exist on the table.
- After defining the proxy table, issue an **update statistics** command for the table. This allows the query optimizer to make intelligent choices regarding join order.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create proxy_table permission defaults to the table owner and is not transferable.

See Also

Commands	**create existing table**, **create table**

create role

Function

Creates a user-defined role; specifies the password expiration interval, the minimum password length, and the maximum number of failed logins allowed for a specified role at creation.

Syntax

```
create role role_name [ with passwd "password" ]
   [, "passwd expiration" | "min passwd length" |
   "max failed_logins" ] option_value ] ]
```

Keywords and Options

role_name – is the name of the new role. It must be unique to the server and conform to the rules for identifiers. It cannot be a variable.

with passwd – attaches a password the user must enter to activate the role.

password – is the password to attach to the role. Passwords must be at least 6 characters in length and must conform to the rules for identifiers. You cannot use variables for passwords.

role_name is the name of the new role. It must be unique to the server and conform to the rules for identifiers. It cannot be a variable.

with passwd attaches a password the user must enter to activate the role.

password is the password to attach to the role. Passwords must be at least six characters in length and must conform to the rules for identifiers. You cannot use variables for passwords.

passwd expiration specifies the password expiration interval in days. It can be any value between 0 and 32767, inclusive.

min passwd length specifies the minimum password length required for the specified login.

max failed_logins specifies the number of allowable failed login attempts for the specified login.

option_value specifies the value for **passwd expiration**, **min passwd length**, or **max failed_logins**.

Examples

1. **`create role doctor_role`**

   Creates a role named doctor_role.

2. **`create role doctor_role with passwd "physician"`**

   Creates a role named doctor_role with the password physician.

3. **`create role intern_role, with passwd "temp244", passwd expiration 7`**

   Sets the password expiration for *intern_role*.

4. **`create role intern_role with passwd “temp244”, max failed_logins 20`**

   Sets the maximum number of failed logins allowed for *intern_role*.

5. **`create role intern_role with passwd "temp244", min passwd length 0`**

   Sets the minimum password length for *intern_role*.

Comments

- The **create role** command creates a role with privileges, permissions, and limitations that you design. For more information on how to use **create role**, see the *System Administration Guide*.

  For information on monitoring and limiting access to objects, see the **set role** command.

- Use **create role** from the *master* database.

- Use the **with passwd** *password* clause to attach a password to a role at creation. If you attach a password to the role, the user granted this role must specify the password to activate the role.

  For information on adding a password to a role after creation, see the **alter role** command.

➤ ***Note***

Passwords attached to user-defined roles do not expire.

- Role names must be unique to the server.

- Role names must not be the same as user names. You can create a role with the same name as a user, but when you grant privileges,

Adaptive Server resolves naming conflicts by making the grant to the user instead of the role.

For more information on naming conflicts, see the **grant role** command.

Restrictions

- The maximum number of roles that can be created per server session is 1024. However, 32 roles are reserved for Sybase system roles, such as **sa_role** and **sso_role**. Therefore, the maximum number of user-defined roles that can be created per server session is 992.
- If you create a role with an attached password, a user cannot activate that role by default at login. Do not create a role with an attached password if the user to whom you grant that role needs to activate the role by default at login.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Security Officer to use **create role**.

create role permission is not included in the **grant all** command.

See Also

Commands	**alter role**, **drop role**, **grant**, **revoke**, **set**
System procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**

create rule

Function

Specifies the domain of acceptable values for a particular column or for any column of a user-defined datatype.

Syntax

```
create rule [owner.]rule_name
   as condition_expression
```

Keywords and Options

rule_name – is the name of the new rule. It must conform to the rules for identifiers and cannot be a variable. Specify the owner's name to create another rule of the same name owned by a different user in the current database. The default value for *owner* is the current user.

condition_expression – specifies the conditions that define the rule. It can be any expression that is valid in a **where** clause, and can include arithmetic operators, relational operators, **in**, **like**, **between**, and so on. However, it cannot reference a column or any other database object. Built-in functions that do not reference database objects **can** be included.

A *condition_expression* takes one argument. The argument is prefixed by the @ sign and refers to the value that is entered via the **update** or **insert** command. You can use any name or symbol to represent the value when you write the rule, but the first character must be the @ sign. Enclose character and date constants in quotes, and precede binary constants with "0x".

Examples

1.
```
create rule limit
as @advance < $1000
```

   Creates a rule named *limit* which limits the value of *advance* to less than $1000.

2.
```
create rule pubid_rule
as @pub_id in ('1389', '0736', '0877')
```

   Creates a rule named *pubid_rule* which restricts the values of *pub_id* to 1389, 0736, or 0877.

3. **create rule picture**
   as @value like '_-%[0-9]'

   Creates a rule named *picture* which restricts the value of *value* to always begin with the indicated characters.

Comments

- To hide the text of a rule, use **sp_hidetext**.
- To rename a rule, use **sp_rename**.

Restrictions

- You can create a rule only in the current database.
- Rules do not apply to the data that already exists in the database at the time the rules are created.
- **create rule** statements cannot be combined with other statements in a single batch.
- You cannot bind a rule to a Adaptive Server-supplied datatype or to a column of type *text*, *image*, or *timestamp*.
- You must drop a rule before you create a new one of the same name, and you must unbind a rule before you drop it. Use:

  sp_unbindrule *objname* [, futureonly]

Binding Rules

- Use the system procedure **sp_bindrule** to bind a rule to a column or user-defined datatype. Its syntax is:

  sp_bindrule *rulename*, *objname* [, futureonly]
- A rule that is bound to a user-defined datatype is activated when you insert a value into, or update, a column of that type. Rules do **not** test values inserted into variables of that type.
- The rule must be compatible with the datatype of the column. For example, you cannot use:

  @value like A%

  as a rule for an exact or approximate numeric column. If the rule is not compatible with the column to which it is bound, Adaptive Server generates an error message when it tries to insert a value, not when you bind it.
- You can bind a rule to a column or datatype without unbinding an existing rule.

- Rules bound to columns always take precedence over rules bound to user-defined datatypes, regardless of which rule was most recently bound. The following chart indicates the precedence when binding rules to columns and user-defined datatypes where rules already exist:

Table 6-14: Rule binding precedence

New Rule Bound To	Old Rule Bound to User-Defined Datatype	Old Rule Bound to Column
User-defined datatype	New rule replaces old	No change
Column	New rule replaces old	New rule replaces old

Rules and Nulls

- Rules do not override column definitions. If a rule is bound to a column that allows null values, you can insert NULL into the column, implicitly or explicitly, even though NULL is not included in the text of the rule. For example, if you create a rule specifying "@val in (1,2,3)" or "@amount > 10000", and bind this rule to a table column that allows null values, you can still insert NULL into that column. The column definition overrides the rule.

Getting Information About Rules

- To get a report on a rule, use **sp_help**.
- To display the text of a rule, which is stored in the *syscomments* system table, execute the system procedure **sp_helptext** with the rule name as the parameter.
- After a rule is bound to a particular column or user-defined datatype, its ID is stored in the *syscolumns* or *systypes* system tables.

Defaults and Rules

- If a column has both a default and a rule associated with it, the default must fall within the domain defined by the rule. A default that conflicts with a rule will never be inserted. Adaptive Server generates an error message each time it attempts to insert the default.

Using Integrity Constraints in Place of Rules

- You can also define rules using **check** integrity constraints with the **create table** statement. However, these constraints are specific for that table; you cannot bind them to other tables. See **create table** and **alter table** for information about integrity constraints.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Transact-SQL extension	To create rules using SQL92-compliant syntax, use the **check** clause of the **create table** statement.

Permissions

create rule permission defaults to the Database Owner, who can transfer it to other users.

See Also

Commands	**alter table**, **create default**, **create table**, **drop default**, **drop rule**
System procedures	**sp_bindrule**, **sp_help**, **sp_helptext**, **sp_hidetext**, **sp_rename**, **sp_unbindrule**

create schema

Function

Creates a new collection of tables, views, and permissions for a database user.

Syntax

```
create schema authorization authorization_name
   create_oject_statement
       [ create_object_statement ... ]
   [ permission_statement ... ]
```

Keywords and Options

authorization_name – must be the name of the current user in the database.

create_object_statement – is a **create table** or **create view** statement.

permission_statement – is a **grant** or **revoke** command.

Examples

```
1. create schema authorization pogo
       create table newtitles (
           title_id tid not null,
           title varchar(30) not null)

       create table newauthors (
           au_id id not null,
           au_lname varchar(40) not null,
           au_fname varchar(20) not null)

       create table newtitleauthors (
           au_id id not null,
           title_id tid not null)

       create view tit_auth_view
       as
           select au_lname, au_fname
               from newtitles, newauthors,
                   newtitleauthors
           where
           newtitleauthors.au_id = newauthors.au_id
           and
           newtitleauthors.title_id =
                newtitles.title_id
```

```
grant select on tit_auth_view to public
revoke select on tit_auth_view from churchy
```

Creates the *newtitles*, *newauthors*, *newtitleauthors* tables, the *tit_auth_view* view, and the corresponding permissions.

Comments

- Schemas can be created only in the current database.
- The *authorization_name*, also called the **schema authorization identifier**, must be the name of the current user.
- The user must have the correct command permissions (**create table** and / or **create view**). If the user creates a view on tables owned by another database user, permissions on the view are checked when a user attempts to access data through the view, not when the view is created.
- The **create schema** command is terminated by:
  - The regular command terminator ("go" is the default in **isql**).
  - Any statement other than **create table**, **create view**, **grant**, or **revoke**.
- If any of the statements within a **create schema** statement fail, the entire command is rolled back as a unit, and none of the commands take effect.
- **create schema** adds information about tables, views, and permissions to the system tables. Use the appropriate drop command (**drop table** or **drop view**) to drop objects created with **create schema**. Permissions granted or revoked in a schema can be changed with the standard **grant** and **revoke** commands outside the schema creation statement.

Standards and Compliance

Standard	Compliance Level
SQL92	Entry level compliant

Permissions

create schema can be executed by any user of a database. The user must have permission to create the objects specified in the schema; that is, **create table** and / or **create view** permission.

See Also

Commands	**create table**, **create view**, **grant**, **revoke**
Utility commands	**isql**

create table

Function

Creates new tables and optional integrity constraints; specifies a locking scheme for the table being created; specifies ascending or descending index order when creating referential integrity constraints that depend on indexes; specifies the expected row size, to reduce row forwarding; specifies a ratio of empty pages to be left for each filled page; allows you to map the table to a table, view, or procedure at a remote location.

Syntax

```
create table [database.[owner].]table_name
(column_name datatype
    [default {constant_expression | user | null}]
    {[{identity | null | not null}]
        [off row | in row]
    | [[constraint constraint_name]
        {{unique | primary key}
       [clustered | nonclustered] [asc | desc]
       [with { { fillfactor = pct
                   | max_rows_per_page = num_rows }
                   , reservepagegap = num_pages }]
        [on segment_name]
        | references [[database.]owner.]ref_table
            [(ref_column)]
        | check (search_condition)}]}...
    | [constraint constraint_name]
    {{unique | primary key}
        [clustered | nonclustered]
        (column_name [asc | desc]
            [{, column_name [asc | desc]}...])
       [with { {fillfactor = pct
             | max_rows_per_page = num_rows },
            reservepagegap = num_pages } ]
         [on segment_name]
    |foreign key (column_name [{, column_name}...])
```

```
        references [[database.]owner.]ref_table
            [(ref_column [{, ref_column}...])]
     | check (search_condition) ... }
[{, {next_column | next_constraint}}...])
[lock {datarows | datapages | allpages }]
[with { max_rows_per_page = num_rows ,
        exp_row_size = num_bytes ,
        reservepagegap = num_pages } ]
[on segment_name]
[ [ external table ] at pathname ]
```

Keywords and Options

table_name – is the explicit name of the new table. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

You cannot use a variable for the table name. The table name must be unique within the database and to the owner. If you have **set quoted_identifier on**, you can use a delimited identifier for the table name. Otherwise, it must conform to the rules for identifiers. For more information about valid table names, see "Identifiers" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

You can create a temporary table by preceding the table name with either a pound sign (#) or "tempdb..". For more information, see "Tables Beginning with # (Temporary Tables)" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

You can create a table in a different database, as long as you are listed in the *sysusers* table and have **create table** permission for that database. For example, to create a table called *newtable* in the database *otherdb*:

```
create table otherdb..newtable
```

or:

```
create table otherdb.yourname.newtable
```

column_name – is the name of the column in the table. It must be unique in the table. If you have **set quoted_identifier on**, you can use a delimited identifier for the column. Otherwise, it must conform to the rules for identifiers. For more information about valid

column names, see "Identifiers" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

datatype – is the datatype of the column. System or user-defined datatypes are acceptable. Certain datatypes expect a length, *n*, in parentheses:

```
datatype(n)
```

Others expect a precision, *p*, and scale, *s*:

```
datatype(p,s)
```

See "Datatypes" for more information.

If Java is enabled in the database, *datatype* can be the name of a Java class, either a system class or a user-defined class, that has been installed in the database. Refer to *Java in Adaptive Server Enterprise* for more information.

default – specifies a default value for a column. If you specify a default, and the user does not provide a value for the column when inserting data, Adaptive Server inserts the default value. The default can be a constant expression, **user**, to insert the name of the user who is performing the insert, or **null**, to insert the null value. Adaptive Server generates a name for the default in the form of *tabname_colname_objid*, where *tabname* is the first 10 characters of the table name, *colname* is the first 5 characters of the column name, and *objid* is the object ID number for the default. Defaults declared for columns with the IDENTITY property have no effect on column values.

constant_expression – is a constant expression to use as a default value for the column. It cannot include the name of any columns or other database objects, but can include built-in functions that do not reference database objects. This default value must be compatible with the datatype of the column, or Adaptive Server generates a datatype conversion error when attempting to insert the default.

user | **null** – specifies that Adaptive Server should insert the user name or the null value as the default if the user does not supply a value. For **user**, the datatype of the column must be either *char(30)* or *varchar(30*). For **null**, the column must allow null values.

identity – indicates that the column has the IDENTITY property. Each table in a database can have one IDENTITY column with a type of *numeric* and a scale of 0. IDENTITY columns are not updatable and do not allow nulls.

IDENTITY columns are used to store sequential numbers, such as invoice numbers or employee numbers, that are generated automatically by Adaptive Server. The value of the IDENTITY column uniquely identifies each row in a table.

null | **not null** – specifies Adaptive Server's behavior during data insertion if no default exists.

null specifies that Adaptive Server assigns a null value if a user does not provide a value.

not null specifies that a user must provide a non-null value if no default exists.

If you do not specify **null** or **not null**, Adaptive Server uses **not null** by default. However, you can switch this default using **sp_dboption** to make the default compatible with the SQL standards.

off row | **in row** – specifies whether a Java-SQL column is stored separate from the row (**off row**) or in storage allocated directly in the row (**in row**).

The storage for an in-row column must not exceed 255 bytes. The default value is **off row.**

Refer to *Java in Adaptive Server Enterprise* for more information.

constraint – introduces the name of an integrity constraint.

constraint_name – is the name of the constraint. It must conform to the rules for identifiers and be unique in the database. If you do not specify the name for a referential or check constraint, Adaptive Server generates a name in the form *tabname_colname_objectid* where *tabname* is the first 10 characters of the table name, *colname* is the first 5 characters of the column name, and *objectid* is the object ID number for the constraint. If you do not specify the name for a unique or primary key constraint, Adaptive Server generates a name in the format *tabname_colname_tabindid* where *tabindid* is a string concatenation of the table ID and index ID.

unique – constrains the values in the indicated column or columns so that no two rows have the same value. This constraint creates a unique index that can be dropped only if the constraint is dropped using **alter table.**

primary key – constrains the values in the indicated column or columns so that no two rows have the same value, and so that the value

cannot be NULL. This constraint creates a unique index that can be dropped only if the constraint is dropped using **alter table**.

clustered | **nonclustered** – specifies that the index created by a **unique** or **primary key** constraint is a clustered or nonclustered index. **clustered** is the default for primary key constraints; **nonclustered** is the default for unique constraints. There can be only one clustered index per table. See **create index** for more information.

asc | desc – specifies whether the index created for a constraint is to be created in ascending or descending order for each column. The default is ascending order.

fillfactor – specifies how full Adaptive Server will make each page when it is creating a new index on existing data. The **fillfactor** percentage is relevant only at the time the index is created. As the data changes, the pages are not maintained at any particular level of fullness.

The default for **fillfactor** is 0; this is used when you do not include **with fillfactor** in the **create index** statement (unless the value has been changed with **sp_configure**). When specifying a **fillfactor**, use a value between 1 and 100.

A **fillfactor** of 0 creates clustered indexes with completely full pages and nonclustered indexes with completely full leaf pages. It leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes. There is seldom a reason to change the **fillfactor**.

If the **fillfactor** is set to 100, Adaptive Server creates both clustered and nonclustered indexes with each page 100 percent full. A **fillfactor** of 100 makes sense only for read-only tables—tables to which no additional data will ever be added.

fillfactor values smaller than 100 (except 0, which is a special case) cause Adaptive Server to create new indexes with pages that are not completely full. A **fillfactor** of 10 might be a reasonable choice if you are creating an index on a table that will eventually hold a great deal more data, but small **fillfactor** values cause each index (or index and data) to take more storage space.

If Component Integration Services is enabled, you cannot use **fillfactor** for remote servers.

◆ ***WARNING!***

Creating a clustered index with a fillfactor affects the amount of storage space your data occupies, since Adaptive Server redistributes the data as it creates the clustered index.

max_rows_per_page – limits the number of rows on data pages and the leaf level pages of indexes. Unlike **fillfactor**, the **max_rows_per_page** value is maintained when data is inserted or deleted.

If you do not specify a value for **max_rows_per_page**, Adaptive Server uses a value of 0 when creating the table. Values for tables and clustered indexes are between 0 and 256. The maximum number of rows per page for nonclustered indexes depends on the size of the index key; Adaptive Server returns an error message if the specified value is too high.

A **max_rows_per_page** of 0 creates clustered indexes with full data pages and nonclustered indexes with full leaf pages. It leaves a comfortable amount of space within the index B-tree in both clustered and nonclustered indexes.

Using low values for **max_rows_per_page** reduces lock contention on frequently accessed data. However, using low values also causes Adaptive Server to create new indexes with pages that are not completely full, uses more storage space, and may cause more page splits.

If Component Integration Services is enabled, you cannot use **max_rows_per_page** for remote servers.

on *segment_name* – specifies that the index is to be created on the named segment. Before the **on** *segment_name* option can be used, the device must be initialized with **disk init**, and the segment must be added to the database with the **sp_addsegment** system procedure. See your System Administrator or use **sp_helpsegment** for a list of the segment names available in your database.

If you specify **clustered** and use the **on** *segment_name* option, the entire table migrates to the segment you specify, since the leaf level of the index contains the actual data pages.

references – specifies a column list for a referential integrity constraint. You can specify only one column value for a column-constraint. By including this constraint with a table that references another table, any data inserted into the **referencing** table must already exist in the **referenced** table.

To use this constraint, you must have **references** permission on the referenced table. The specified columns in the referenced table must be constrained by a unique index (created by either a **unique** constraint or a **create index** statement). If no columns are specified, there must be a **primary key** constraint on the appropriate columns in the referenced table. Also, the datatypes of the referencing table columns must match the datatype of the referenced table columns.

foreign key – specifies that the listed column(s) are foreign keys in this table whose target keys are the columns listed in the following **references** clause. The foreign key syntax is permitted only for table-level constraints, not for column-level constraints.

ref_table – is the name of the table that contains the referenced columns. You can reference tables in another database. Constraints can reference up to 192 user tables and internally generated worktables.

ref_column – is the name of the column or columns in the referenced table.

check – specifies a *search_condition* constraint that Adaptive Server enforces for all the rows in the table. You can specify **check** constraints as table or column constraints; **create table** allows multiple **check** constraints in a column definition.

search_condition – is the **check** constraint on the column values. These constraints can include:

- A list of constant expressions introduced with **in**
- A set of conditions introduced with **like**, which may contain wildcard characters

Column and table check constraints can reference any columns in the table.

An expression can include arithmetic operators and functions. The *search_condition* cannot contain subqueries, aggregate functions, host variables, or parameters.

next_column | *next_constraint* – indicates that you can include additional column definitions or table constraints (separated by commas) using the same syntax described for a column definition or table constraint definition.

lock datarows | datapages | allpages – specifies the locking scheme to be used for the table. The default is the server-wide setting for the configuration parameter **lock scheme**.

exp_row_size = *num_bytes* – specifies the expected row size; applies only to **datarows** and **datapages** locking schemes, and only to tables with variable-length rows. Valid values are 0, 1, and any value between the minimum and maximum row length for the table. The default value is 0, which means a server-wide setting is applied.

reservepagegap = *num_pages* – specifies the ratio of filled pages to empty pages that are to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

external table – specifies that the object is a remote table or view. **external table** is the default, so specifying this is optional.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object;aux1.aux2

where:

- *server_name* (required) is the name of the server that contains the remote object.
- *dbname* (optional) is the name of the database managed by the remote server that contains this object.
- *owner* (optional) is the name of the remote server user that owns the remote object.
- *object* (required) is the name of the remote table or view.
- *aux1.aux2* (optional) is a string of characters that is passed to the remote server during a **create table** or **create index** command. This string is used only if the server is class *db2*. *aux1* is the DB2 database in which to place the table, and *aux2* is the DB2 table space in which to place the table.

on *segment_name* – specifies the name of the segment on which to place the table. When using **on** *segment_name*, the logical device must already have been assigned to the database with **create database** or **alter database**, and the segment must have been created in the database with **sp_addsegment**. See your System

Administrator or use **sp_helpsegment** for a list of the segment names available in your database.

Examples

1.
```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null,
price money null,
advance money null,
total_sales int null,
notes varchar(200)null,
pubdate datetime not null,
contract bit not null)
```

   Creates the *titles* table.

2.
```
create table "compute"
("max" int, "min" int, "total score" int)
```

   Creates the *compute* table. The table name and the column names, *max* and *min*, are enclosed in double quotes because they are reserved words. The *total score* column name is enclosed in double quotes because it contains an embedded blank. Before creating this table, you must **set quoted_identifier on**.

3.
```
create table sales
(stor_id         char(4)      not null,
ord_num          varchar(20)  not null,
date             datetime     not null,
unique clustered (stor_id, ord_num))
```

   Creates the *sales* table and a clustered index in one step with a unique constraint. (In the *pubs2* database installation script, there are separate **create table** and **create index** statements.)

4.
```
create table salesdetail
(stor_id    char(4)                 not null,
ord_num     varchar(20)             not null,
title_id     tid                    not null
             references titles(title_id),
qty          smallint default 0     not null,
discount    float                   not null,

constraint salesdet_constr
    foreign key (stor_id, ord_num)
    references sales(stor_id, ord_num))
```

Creates the *salesdetail* table with two referential integrity constraints and one default value. There is a table-level, referential integrity constraint named *salesdet_constr* and a column-level, referential integrity constraint on the *title_id* column without a specified name. Both constraints specify columns that have unique indexes in the referenced tables (*titles* and *sales*). The **default** clause with the *qty* column specifies 0 as its default value.

5.
```
create table publishers
(pub_id char(4) not null
   check (pub_id in ("1389", "0736", "0877",
       "1622", "1756")
    or pub_id like "99[0-9][0-9]"),
pub_name     varchar(40)  null,
city     varchar(20)  null,
state     char(2)  null)
```

Creates the table *publishers* with a check constraint on the *pub_id* column. This column-level constraint can be used in place of the *pub_idrule* included in the *pubs2* database:

```
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877",
    "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

6.
```
create table sales_daily
(stor_id         char(4)       not null,
ord_num          numeric(10,0)   identity,
ord_amt          money         null)
```

Specifies the *ord_num* column as the IDENTITY column for the *sales_daily* table. The first time you insert a row into the table, Adaptive Server assigns a value of 1 to the IDENTITY column. On each subsequent insert, the value of the column increments by 1.

```
7. create table new_titles (
        title_id    tid,
        title       varchar(80) not null,
        type        char(12) ,
        pub_id      char(4) null,
        price       money null,
        advance     money null,
        total_sales int null,
        notes       varchar(200) null,
        pubdate     datetime,
        contract    bit    )
   lock datapages
   with exp_row_size = 200
```

Specifies the **datapages** locking scheme for the *new_titles* table and an expected row size of 200.

```
8. create table new_publishers (
   pub_id          char(4) not null,
   pub_name        varchar(40) null,
   city            varchar(20) null,
   state           char(2) null )
   lock datarows
   with reservepagegap = 16
```

Specifies the **datarows** locking scheme and sets a **reservepagegap** value of 16 so that extent I/O operations leave 1 blank page for each 15 filled pages.

```
9. create table sales_south
   (stor_id         char(4)      not null,
   ord_num          varchar(20)  not null,
   date             datetime     not null,
   unique clustered (stor_id asc, ord_num desc))
```

Creates a constraint supported by a unique clustered index; the index order is ascending for *stor_id* and descending for *ord_num*.

```
10.create table t1
   (a  int,
    b  char(10))
   at "SERVER_A.db1.joe.t1"
```

Creates a table named *t1* at the remote server SERVER_A and creates a proxy table named *t1* that is mapped to the remote table.

```
11.create table employees
   (name varchar(30),
   home_addr Address,
   mailing_addr Address2Line)
```

Creates a table named *employees*. *name* is of type *varchar*, *home_addr* is a Java-SQL column of type **Address**, and *mailing_addr* is a Java-SQL column of type **Address2Line**. Both **Address** and **Address2Line** are Java classes installed in the database.

Comments

- **create table** creates a table and optional integrity constraints. The table is created in the currently open database unless you specify a different database in the **create table** statement. You can create a table or index in another database, if you are listed in the *sysusers* table and have **create table** permission in the database.
- Space is allocated to tables and indexes in increments of one extent, or eight pages, at a time. Each time an extent is filled, another extent is allocated. To see the amount of space allocated and used by a table, use **sp_spaceused**.
- When using **create table** from Component Integration Services with a column defined as *char*(*n*) NULL, Component Integration Services will create the column as *varchar*(*n*) on the remote server.

Restrictions

- There can be up to 2 billion tables per database and 250 user-defined columns per table. The number of rows per table is limited only by available storage.
- The maximum number of bytes per row depends on the locking scheme for the table. The maximum number of bytes for user data is 1960 bytes in an allpages-locked table. For data-only-locked tables, deduct 2 bytes for each variable-length column or column that allows null values.
- If you create tables with *varchar*, *nvarchar*, or *varbinary* columns whose total defined width is greater than the maximum allowed row size, a warning message appears, but the table is created. If you try to insert more than the maximum number bytes into such a row, or to **update** a row so that its total row size is greater than the maximum length, Adaptive Server produces an error message, and the command fails.

➤ ***Note***

When a **create table** command occurs within an **if...else** block or a **while** loop, Adaptive Server creates the schema for the table before determining whether the condition is true. This may lead to errors if the table already exists. Make sure a table with the same name does not already exist in the database.

Column Definitions

- When you create a column from a user-defined datatype:
  - You cannot change the length, precision, or scale.
  - You can use a NULL type to create a NOT NULL column, but not to create an IDENTITY column.
  - You can use a NOT NULL type to create a NULL column or an IDENTITY column.
  - You can use an IDENTITY type to create a NOT NULL column, but the column inherits the IDENTITY property. You cannot use an IDENTITY type to create a NULL column.
- Only columns with variable-length datatypes can store null values. When you create a NULL column with a fixed-length datatype, Adaptive Server automatically converts it to the corresponding variable-length datatype. Adaptive Server does not inform the user of the type change.

  The following table lists the fixed-length datatypes and the variable-length datatypes to which they are converted. Certain variable-length datatypes, such as *moneyn*, are reserved types that cannot be used to create columns, variables, or parameters:

Table 6-15: Variable-length datatypes used to store nulls

Original Fixed-Length Datatype	Converted To
char	*varchar*
nchar	*nvarchar*
binary	*varbinary*
datetime	*datetimn*
float	*floatn*
int, *smallint*, and *tinyint*	*intn*
decimal	*decimaln*
numeric	*numericn*
money and *smallmoney*	*moneyn*

- You can create column defaults in two ways: by declaring the default as a column constraint in the **create table** or **alter table** statement, or by creating the default using the **create default** statement and binding it to a column using **sp_bindefault**.
- For a report on a table and its columns, execute the system procedure **sp_help**.

Temporary Tables

- Temporary tables are stored in the temporary database, *tempdb*.
- The first 13 characters of a temporary table name must be unique per session. Such tables can be accessed only by the current Adaptive Server session. They are stored in *tempdb..objects* by their names plus a system-supplied numeric suffix, and they disappear at the end of the current session or when they are explicitly dropped.
- Temporary tables created with the "tempdb.." prefix are shareable among Adaptive Server user sessions. They exist until they are explicitly dropped by their owner or until Adaptive Server reboots. Create temporary tables with the "tempdb.." prefix from inside a stored procedure only if you intend to share the table among users and sessions. To avoid inadvertent sharing of temporary tables, use the # prefix when creating and dropping temporary tables in stored procedures.
- Temporary tables can be used by multiple users during an Adaptive Server session. However, the specific user session usually cannot be identified because temporary tables are created with the "guest" user ID of 2. If more than one user runs the process that creates the temporary table, each user is a "guest" user so the *uid* values are all the same. Therefore, there is no way to know which user session in the temporary table is for a specific user. It is possible that the SA can add the user to the temporary table using **sp_addlogin**, in which case the individual *uid* would be available for that user's session in the temporary table, but this circumstance is unlikely.
- You can associate rules, defaults and indexes with temporary tables, but you cannot create views on temporary tables or associate triggers with them.
- When you create a temporary table, you can use a user-defined datatype only if the type is in *tempdb..systypes*. To add a user-defined datatype to *tempdb* for the current session only, execute **sp_addtype** while using *tempdb*. To add the datatype permanently,

execute **sp_addtype** while using *model*, then restart Adaptive Server so that *model* is copied to *tempdb*.

Using Indexes

- A table "follows" its clustered index. If you create a table on one segment, and then create its clustered index on another segment, the table migrates to the segment where the index is created.
- You can make inserts, updates, and selects faster by creating a table on one segment and its nonclustered indexes on another segment, if the segments are on separate physical devices. For more information, see the *Performance and Tuning Guide*.

Renaming a Table or Its Columns

- Use **sp_rename** to rename a table or column.
- After renaming a table or any of its columns, use **sp_depends** to determine which procedures, triggers, and views depend on the table, and redefine these objects.

◆ ***WARNING!***

If you do not redefine these dependent objects, they will no longer work after Adaptive Server recompiles them.

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing.

Defining Integrity Constraints

- The **create table** statement helps control a database's integrity through a series of integrity constraints as defined by the SQL standards. These integrity constraint clauses restrict the data that users can insert into a table. You can also use defaults, rules, indexes, and triggers to enforce database integrity.

  Integrity constraints offer the advantages of defining integrity controls in one step during the table creation process and of simplifying the process to create those integrity controls.

However, integrity constraints are more limited in scope and less comprehensive than defaults, rules, indexes, and triggers.

- You must declare constraints that operate on more than one column as table-level constraints; declare constraints that operate on just one column as column-level constraints. The difference is syntactic: you place column-level constraints after the column name and datatype, before the delimiting comma (see example 5). You enter table-level constraints as separate comma-delimited clauses (see example 4). Adaptive Server treats table-level and column-level constraints the same way; neither way is more efficient than the other.
- You can create the following types of constraints at the table level or the column level:
  - A **unique** constraint requires that no two rows in a table have the same values in the specified columns. In addition, a **primary key** constraint requires that there be no null values in the column.
  - A referential integrity (**references**) constraint requires that the data being inserted or updated in specific columns has matching data in the specified table and columns.
  - A **check** constraint limits the values of the data inserted into the columns.

  You can also enforce data integrity by restricting the use of null values in a column (the **null** or **not null** keywords) and by providing default values for columns (the **default** clause).
- You can use the system procedures **sp_primarykey**, **sp_foreignkey**, and **sp_commonkey** to save information in system tables, which can help clarify the relationships between tables in a database. These system procedures do not enforce the key relationships or duplicate the functions of the **primary key** and **foreign key** keywords in a **create table** statement. For a report on keys that have been defined, use **sp_helpkey**. For a report on frequently used joins, execute **sp_helpjoins**.
- Transact-SQL provides several mechanisms for integrity enforcement. In addition to the constraints you can declare as part of **create table**, you can create rules, defaults, indexes, and

triggers. The following table summarizes the integrity constraints and describes the other methods of integrity enforcement:

Table 6-16: Methods of integrity enforcement

In *create table*	Other Methods
unique constraint	**create unique index** (on a column that allows null values)
primary key constraint	**create unique index** (on a column that does not allow null values)
references constraint	**create trigger**
check constraint (table level)	**create trigger**
check constraint (column level)	**create trigger** or **create rule** and **sp_bindrule**
default clause	**create default** and **sp_bindefault**

Which method you choose depends on your requirements. For example, triggers provide more complex handling of referential integrity (such as referencing other columns or objects) than those declared in **create table**. Also, the constraints defined in a **create table** statement are specific for that table; unlike rules and defaults, you cannot bind them to other tables, and you can only drop or change them using **alter table**. Constraints cannot contain subqueries or aggregate functions, even on the same table.

- The **create table** command can include many constraints, with these limitations:
  - The number of **unique** constraints is limited by the number of indexes that table can have.
  - A table can have only one **primary key** constraint.
  - You can include only one **default** clause per column in a table, but you can define different constraints on the same column.

  For example:

```
create table discount_titles
(title_id   varchar(6)   default "PS7777" not null
        unique clustered
        references titles(title_id)
        check (title_id like "PS%"),
             new_price   money)
```

  Column *title_id* of the new table *discount_titles* is defined with each integrity constraint.

- You can create error messages and bind them to referential integrity and **check** constraints. Create messages with **sp_addmessage** and bind them to the constraints with **sp_bindmsg**. For more information, see **sp_addmessage** and **sp_bindmsg**.
- Adaptive Server evaluates check constraints before enforcing the referential constraints, and evaluates triggers after enforcing all the integrity constraints. If any constraint fails, Adaptive Server cancels the data modification statement; any associated triggers do not execute. However, a constraint violation **does not** roll back the current transaction.
- In a referenced table, you cannot update column values or delete rows that match values in a referencing table. Update or delete from the referencing table first, then try updating or deleting from the referenced table.
- You must drop the referencing table before you drop the referenced table; otherwise, a constraint violation will occur.
- For information about constraints defined for a table, use **sp_helpconstraint**.

Unique and Primary Key Constraints

- You can declare **unique** constraints at the column level or the table level. **unique** constraints require that all values in the specified columns be unique. No two rows in the table can have the same value in the specified column.
- A **primary key** constraint is a more restrictive form of **unique** constraint. Columns with **primary key** constraints cannot contain null values.

➤ ***Note***

The **create table** statement's **unique** and **primary key** constraints create indexes that define unique or primary key attributes of columns. **sp_primarykey**, **sp_foreignkey**, and **sp_commonkey** define logical relationships between columns. These relationships must be enforced using indexes and triggers.

- Table-level **unique** or **primary** key constraints appear in the **create table** statement as separate items and must include the names of one or more columns from the table being created.

- **unique** or **primary key** constraints create a unique index on the specified columns. The **unique** constraint in example 3 creates a unique, clustered index, as does the statement:

  ```
  create unique clustered index salesind
      on sales (stor_id, ord_num)
  ```

  The only difference is the index name, which you could set to *salesind* by naming the constraint.

- The definition of **unique** constraints in the SQL standards specifies that the column definition cannot allow null values. By default, Adaptive Server defines the column as not allowing null values (if you have not changed this using **sp_dboption**) when you omit **null** or **not null** in the column definition. In Transact-SQL, you can define the column to allow null values along with the **unique** constraint, since the unique index used to enforce the constraint allows you to insert a null value.

- **unique** constraints create unique, nonclustered indexes by default; **primary key** constraints create unique, clustered indexes by default. There can be only one clustered index on a table, so you can specify only one **unique clustered** or **primary key clustered** constraint.

- The **unique** and **primary key** constraints of **create table** offer a simpler alternative to the **create index** statement. However, they have the following limitations:

  - You cannot create nonunique indexes.
  - You cannot use all the options provided by **create index**.
  - You must drop these indexes using **alter table drop constraint.**

Referential Integrity Constraints

- Referential integrity constraints require that data inserted into a **referencing** table that defines the constraint must have matching values in a **referenced** table. A referential integrity constraint is satisfied for either of the following conditions:

  - The data in the constrained column(s) of the referencing table contains a null value.
  - The data in the constrained column(s) of the referencing table matches data values in the corresponding columns of the referenced table.

  Using the *pubs2* database as an example, a row inserted into the *salesdetail* table (which records the sale of books) must have a valid *title_id* in the *titles* table. *salesdetail* is the referencing table

and *titles* table is the referenced table. Currently, *pubs2* enforces this referential integrity using a trigger. However, the *salesdetail* table could include this column definition and referential integrity constraint to accomplish the same task:

```
title_id tid
    references titles(title_id)
```

- The maximum number of table references allowed for a query is 192. Use the system procedure **sp_helpconstraint** to check a table's referential constraints.
- A table can include a referential integrity constraint on itself. For example, the *store_employees* table in *pubs3*, which lists employees and their managers, has the following self-reference between the *emp_id* and *mgr_id* columns:

  ```
  emp_id id primary key,
  mgr_id id null
        references store_employees(emp_id),
  ```

  This constraint ensures that all managers are also employees, and that all employees have been assigned a valid manager.
- You cannot drop the referenced table until the referencing table is dropped or the referential integrity constraint is removed (unless it includes only a referential integrity constraint on itself).
- Adaptive Server does not enforce referential integrity constraints for temporary tables.
- To create a table that references another user's table, you must have **references** permission on the referenced table. For information about assigning **references** permissions, see the **grant** command.
- Table-level, referential integrity constraints appear in the **create table** statement as separate items. They must include the **foreign key** clause and a list of one or more column names.

  Column names in the **references** clause are optional only if the columns in the referenced table are designated as a primary key through a **primary key** constraint.

  The referenced columns must be constrained by a unique index in that referenced table. You can create that unique index using either the **unique** constraint or the **create index** statement.
- The datatypes of the referencing table columns must match the datatypes of the referenced table columns. For example, the datatype of *col1* in the referencing table (*test_type*) matches the datatype of *pub_id* in the referenced table (*publishers*):

```
create table test_type
(col1 char(4) not null
    references publishers(pub_id),
col2 varchar(20) not null)
```

- The referenced table must exist at the time you define the referential integrity constraint. For tables that cross-reference one another, use the **create schema** statement to define both tables simultaneously. As an alternative, create one table without the constraint and add it later using **alter table**. See **create schema** or **alter table** for more information.
- The **create table** referential integrity constraints offer a simple way to enforce data integrity. Unlike triggers, they **cannot**:
  - Cascade changes through related tables in the database
  - Enforce complex restrictions by referencing other columns or database objects
  - Perform "what-if" analysis

  Referential integrity constraints do not roll back transactions when a data modification violates the constraint. Triggers allow you to choose whether to roll back or continue the transaction depending on how you handle referential integrity.

➤ ***Note***

Adaptive Server checks referential integrity constraints before it checks any triggers, so a data modification statement that violates the constraint does not also fire the trigger.

Using Cross-Database Referential Integrity Constraints

- When you create a cross-database constraint, Adaptive Server stores the following information in the *sysreferences* system table of each database:

Table 6-17: Information stored for referential integrity constraints

Information Stored in *sysreferences*	Columns with Information About the Referenced Table	Columns with Information About the Referencing Table
Key column IDs	*refkey1* through *refkey16*	*fokey1* through *fokey16*
Table ID	*reftabid*	*tableid*

Table 6-17: Information stored for referential integrity constraints

Information Stored in *sysreferences*	Columns with Information About the Referenced Table	Columns with Information About the Referencing Table
Database ID	*pmrydbid*	*frgndbid*
Database name	*pmrydbname*	*frgndbname*

- You can drop the referencing table or its database without problems. Adaptive Server automatically removes the foreign key information from the referenced database.
- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains the referenced table, or
  - Rename either database with **sp_renamedb**.

  You must remove the cross-database constraint with **alter table** before you can do any of these actions.
- Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of databases containing cross-database constraints could cause database corruption.

- The *sysreferences* system table stores the **name** and the ID number of the external database. Adaptive Server cannot guarantee referential integrity if you use **load database** to change the database name or to load it onto a different server.

◆ ***WARNING!***

Before dumping a database in order to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

check Constraints

- A **check** constraint limits the values a user can insert into a column in a table. A **check** constraint specifies a *search_condition* that any non-null value must pass before it is inserted into the table. A *search_condition* can include:
  - A list of constant expressions introduced with **in**
  - A range of constant expressions introduced with **between**
  - A set of conditions introduced with **like**, which can contain wildcard characters

  An expression can include arithmetic operators and Transact-SQL built-in functions. The *search_condition* cannot contain subqueries, aggregate functions, or a host variable or parameter. Adaptive Server does not enforce **check** constraints for temporary tables.
- If the **check** constraint is a column-level **check** constraint, it can reference only the column in which it is defined; it cannot reference other columns in the table. Table-level **check** constraints can reference any column in the table.
- **create table** allows multiple **check** constraints in a column definition.
- **check** integrity constraints offer an alternative to using rules and triggers. They are specific to the table in which they are created, and cannot be bound to columns in other tables or to user-defined datatypes.
- **check** constraints do not override column definitions. If you declare a **check** constraint on a column that allows null values, you can insert NULL into the column, implicitly or explicitly, even though NULL is not included in the *search_condition*. For example, if you create a **check** constraint specifying "pub_id in ("1389", "0736", "0877", "1622", "1756")" or "@amount > 10000" in a table column that allows null values, you can still insert NULL into that column. The column definition overrides the **check** constraint.

IDENTITY Columns

- The first time you insert a row into the table, Adaptive Server assigns the IDENTITY column a value of 1. Each new row gets a column value that is 1 higher than the last value. This value takes precedence over any defaults declared for the column in the **create table** statement or bound to the column with the **sp_bindefault**

system procedure. The maximum value that can be inserted into the IDENTITY column is 10 PRECISION - 1.

- Inserting a value into the IDENTITY column allows you to specify a seed value for the column or to restore a row that was deleted in error. The table owner, Database Owner, or System Administrator can explicitly insert a value into an IDENTITY column after using **set identity_insert** *table_name* **on** for the base table. Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the value. You can insert any positive integer.
- You can reference an IDENTITY column using the **syb_identity** keyword, qualified by the table name where necessary, in place of the actual column name.
- System Administrators can use the **auto identity** database option to automatically include a 10-digit IDENTITY column in new tables. To turn on this feature in a database, use:

  ```
  sp_dboption database_name, "auto identity", "true"
  ```

  Each time a user creates a table in the database without specifying either a **primary** key, a **unique** constraint, or an IDENTITY column, Adaptive Server automatically defines an IDENTITY column. This column, SYB_IDENTITY_COL, is not visible when you retrieve columns with the **select** * statement. You must explicitly include the column name in the select list.
- Server failures can create gaps in IDENTITY column values. The maximum size of the gap depends on the setting of the **identity burning set factor** configuration parameter. Gaps can also occur due to transaction rollbacks, the deletion of rows, or the manual insertion of data into the IDENTITY column.

Specifying a Locking Scheme

- To specify the locking scheme for a table, use the keyword **lock** and one of the following locking schemes:
  - **allpages** locking, which locks data pages and the indexes affected by queries
  - **datapages** locking, which locks only data pages
  - **datarows** locking, which locks only data rows

  If you do not specify a locking scheme, the default locking scheme for the server is used. The server-wide default is set with the configuration parameter **lock scheme**.

- The locking scheme for a table can be changed with the **alter table** command.

Space Management Properties

- The space management properties **fillfactor**, **max_rows_per_page**, **exp_row_size**, and **reservepagegap** help manage space usage for tables in the following ways:
  - **fillfactor** leaves extra space on pages when indexes are created, but the **fillfactor** is not maintained over time.
  - **max_rows_per_page** limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables, since reducing the number of rows can reduce lock contention. If you specify a **max_rows_per_page** value and **datapages** or **datarows** locking, a warning message is printed. The table is created, and the value is stored in *sysindexes*, but it is applied only if the locking scheme is changed later to **allpages**.
  - **exp_row_size** specifies the expected size of a data row. It applies only to data rows, not to indexes, and applies only to data-only-locked tables that have variable-length columns. It is used to reduce the number of forwarded rows in data-only-locked tables. It is needed mainly for tables where rows have null or short columns when first inserted, bus increase in size as a result of subsequent updates. **exp_row_size** reserves space on the data page for the row to grow to the specified size. If you specify **exp_row_size** when you create an allpages-locked table, a warning message is printed. The table is created, and the value is stored in *sysindexes*, but it is only applied if the locking scheme is changed later to **datapages** or **datarows**.
  - **reservepagegap** specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to both data and index pages, in all locking schemes.
- Table 6-18 shows the valid combinations of space management properties and locking scheme. If a **create table** command includes incompatible combinations, a warning message is printed and the table is created. The values are stored in system tables, but are

not applied. If the locking scheme for a table changes so that the properties become valid, then they are used.

Table 6-18: Space management properties and locking schemes

Property	allpages	datapages	datarows
max_rows_per_page	Yes	No	No
exp_row_size	No	Yes	Yes
reservepagegap	Yes	Yes	Yes
fillfactor	Yes	Yes	Yes

- Table 6-19 shows the default values and the effects of using default values for the space management properties.

Table 6-19: Defaults and effects of space management properties

Property	Default	Effect of Using the Default
max_rows_per_page	0	Fits as many rows as possible on the page, up to a maximum of 255
exp_row_size	0	Uses the server-wide default value, set with the configuration parameter **default exp_row_size percent**
reservepagegap	0	Leaves no empty pages during extent allocations
fillfactor	0	Fully packs leaf pages, with space left on index pages

Using *exp_row_size*

- If an application inserts short rows into a data-only-locked table and updates them later so that their length increases, use **exp_row_size** to reduce the number of times that rows in data-only-locked tables are forwarded to new locations.

Using *reservepagegap*

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The **reservepagegap** keyword causes these commands to leave empty pages so that subsequent page allocations will take place close to

the page being split or close to the page from which a row is being forwarded. Table 6-20 shows when **reservepagegap** is applied.

Table 6-20: When reservepagegap is applied

Command	Applies to Data Pages	Applies to Index Pages
Fast **bcp**	Yes	Fast **bcp** is not used if indexes exist.
Slow **bcp**	Only for heap tables, not for tables with a clustered index	Extent allocation not performed
select into	Yes	No indexes exist on the target table
create index or **alter table...constraint**	Yes, for clustered indexes	Yes
reorg rebuild	Yes	Yes
alter table...lock (For allpages-locking to data-only-locking, or vice versa)	Yes	Yes

- The **reservepagegap** value for a table is stored in *sysindexes* and is applied when any of the above operations on a table are executed. To change the stored value, use the system procedure **sp_chgattribute.**
- **reservepagegap** is not applied to worktables or sorts on worktables.

Using *at*

- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.

Java-SQL Columns

- If Java is enabled in the database, you can creates tables with Java-SQL columns. Refer to *Java in Adaptive Server Enterprise* for detailed information.
- The declared class (*datatype*) of the Java-SQL column must implement either the **Serializable** or **Externalizable** interface.
- When you create a table, a Java-SQL column cannot be specified:
  - As a foreign key

  - In a references clause
  - As having the UNIQUE property
  - As the primary key
- If **in row** is specified, then the value stored cannot exceed 255 bytes.
- If **off row** is specified, then:
  - The column cannot be referenced in a check constraint.
  - The column cannot be referenced in a **select** that specifies **distinct**.
  - The column cannot be specified in a comparison operator, in a predicate, or in a **group by** clause.

Getting Information on Tables

- **sp_help** displays information about tables, listing any attributes (such as cache bindings) assigned to the specified table and its indexes, giving the attribute's class, name, integer value, character value, and comments.
- **sp_depends** displays information about the view(s), trigger(s), and procedure(s) in the database that depend on a table.
- **sp_helpindex** reports information about the indexes created on a table.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The following are Transact-SQL extensions: • Use of a database name to qualify a table or column name • IDENTITY columns • The **not null** column default • The **asc** and **desc** options • The **reservepagegap** option • The **lock** clause • The **on *segment_name*** clause See "System and User-Defined Datatypes" or datatype compliance information.

Permissions

create table permission defaults to the Database Owner, who can transfer it to other users. Any user can create temporary tables.

See Also

Commands	**alter table**, **create existing table**, **create index**, **create rule**, **create schema**, **create view**, **drop index**, **drop rule**, **drop table**
System procedures	**sp_addmessage**, **sp_addsegment**, **sp_addtype**, **sp_bindmsg**, **sp_chgattribute**, **sp_commonkey**, **sp_depends**, **sp_foreignkey**, **sp_help**, **sp_helpjoins**, **sp_helpsegment**, **sp_primarykey**, **sp_rename**, **sp_spaceused**

create trigger

Function

Creates a trigger, a type of stored procedure that is often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.

Syntax

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   for {insert , update , delete}
   as SQL_statements
```

Or, using the **if update** clause:

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   for {insert , update}
   as
       [if update (column_name)
           [{and | or} update (column_name)]...]
           SQL_statements
       [if update (column_name)
           [{and | or} update (column_name)]...
           SQL_statements]...
```

Keywords and Options

trigger_name – is the name of the trigger. It must conform to the rules for identifiers and be unique in the database. Specify the owner's name to create another trigger of the same name owned by a different user in the current database. The default value for *owner* is the current user. If you use an owner name to qualify a trigger, you must explicitly qualify the table name the same way.

You cannot use a variable for a trigger name.

table_name – is the name of the table on which to create the trigger. If more than one table of the same name exists in the database, specify the owner's name. The default value for *owner* is the current user.

insert, **update**, **delete** – can be included in any combination. **delete** cannot be used with the **if update** clause.

SQL_statements – specify trigger conditions and trigger actions. Trigger conditions determine whether the attempted **insert**, **update**, or **delete** causes the trigger actions to be carried out. The SQL statements often include a subquery preceded by the keyword **if**. In example 2, below, the subquery that follows the keyword **if** is the trigger condition.

Trigger actions take effect when the user action (**insert**, **update**, or **delete**) is attempted. If multiple trigger actions are specified, they are grouped with **begin** and **end**.

See "Triggers and Transactions" for a list of statements that are not allowed in a trigger definition. See "The deleted and inserted Logical Tables" for information about the *deleted* and *inserted* logical tables that can be included in trigger definitions.

if update – is used to test whether the specified column is included in the **set** list of an **update** statement or is affected by an **insert**. This allows specified trigger actions to be associated with updates to specified columns (see example 3). More than one column can be specified, and you can use more than one **if update** statement in a **create trigger** statement (see example 5).

Examples

1.
```
create trigger reminder
on titles
for insert, update as
print "Don't forget to print a report for
accounting."
```

   Prints a message when anyone tries to add data or change data in the *titles* table.

2.
```
create trigger t1
on titleauthor
for insert as
if (select count(*)
     from titles, inserted
     where titles.title_id = inserted.title_id) = 0
begin
print "Please put the book's title_id in the
        titles table first."
rollback transaction
end
```

   Prevents insertion of a new row into *titleauthor* if there is no corresponding *title_id* in the *titles* table.

3.
```
create trigger t2
on publishers
for update as
if update (pub_id) and @@rowcount = 1
begin
    update titles
    set titles.pub_id = inserted.pub_id
    from titles, deleted, inserted
    where deleted.pub_id = titles.pub_id
end
```

   If the *pub_id* column of the *publishers* table is changed, make the corresponding change in the *titles* table.

4.
```
create trigger t3
on titleauthor
for delete as
begin
    delete titles
    from titles, deleted
    where deleted.title_id = titles.title_id
    delete titleauthor
    from titleauthor, deleted
    where deleted.title_id = titleauthor.title_id
    print "All references to this title have been
    deleted from titles and titleauthor."
end
```

   Deletes title from the *titles* table if any row is deleted from *titleauthor*. If the book was written by more than one author, other references to it in *titleauthor* are also deleted.

5.
```
create trigger stopupdatetrig
on titles
for update
as
if update (title_id)
  and datename(dw, getdate())
  in ("Saturday", "Sunday")
  begin
    rollback transaction
    print "We don't allow changes to"
    print "primary keys on the weekend!"
  end
```

```
if update (price) or update (advance)
  if (select count(*) from inserted
    where (inserted.price * inserted.total_sales)
    < inserted.advance) > 0
    begin
      rollback transaction
      print "We don't allow changes to price or"
      print "advance for a title until its total"
      print "revenue exceeds its latest advance."
    end
```

Prevents updates to the primary key on weekends. Prevents updates to the price or advance of a title unless the total revenue amount for that title surpasses its advance amount.

Comments

- A trigger fires only once per data modification statement. A complex query containing a **while** loop may repeat an **update** or **insert** many times, and the trigger is fired each time.

Triggers and Referential Integrity

- Triggers are commonly used to enforce **referential integrity** (integrity rules about relationships between the primary and foreign keys of tables or views), to supply cascading deletes, and to supply cascading updates (see examples 2, 3, and 4, respectively).
- A trigger fires only after the data modification statement has completed and Adaptive Server has checked for any datatype, rule, or integrity constraint violations. The trigger and the statement that fires it are treated as a single transaction that can be rolled back from within the trigger. If a severe error is detected, the entire transaction is rolled back.
- You can also enforce referential integrity using constraints defined with the **create table** statement as an alternative to using **create trigger.** See **create table** and **alter table** for information about integrity constraints.

The *deleted* and *inserted* Logical Tables

- *deleted* and *inserted* are logical (conceptual) tables. They are structurally like the table for which the trigger is defined—that is, the table on which the user action is attempted—and hold the old values or new values of the rows that would be changed by the user action.

- *deleted* and *inserted* tables can be examined by the trigger to determine whether or how the trigger action(s) should be carried out, but the tables themselves cannot be altered by the trigger's actions.
- *deleted* tables are used with **delete** and **update**; *inserted* tables, with **insert** and **update**. (An update is a delete followed by an insert: it affects the *deleted* table first, and then the *inserted* table).

Trigger Restrictions

- You can create a trigger only in the current database. If you use an owner name to qualify a trigger, you must explicitly qualify the table name the same way. A trigger can reference objects outside the current database.
- A trigger cannot apply to more than one table. However, the same trigger action can be defined for more than one user action (for example, **insert** and **update**) in the same **create trigger** statement. A table can have a maximum of three triggers—one each for **insert**, **update**, and **delete**.
- Each new trigger in a table or column for the same operation (**insert**, **update**, or **delete**) overwrites the previous one. No warning message is given before the overwrite occurs.
- You cannot create a trigger on a temporary table.
- You cannot create a trigger on a view.
- You cannot create a trigger on a system table.
- You cannot use triggers that select from a *text* or *image* column of the inserted or deleted table.
- It is recommended that a trigger not include **select** statements that return results to the user, since special handling for these returned results would have to be written into every application program that allows modifications to the trigger table.
- If a trigger references table names, column names, or view names that are not valid identifiers, you must **set quoted_identifier on** before the **create trigger** command and enclose each such name in double quotes. The **quoted_identifier** option does **not** need to be on when the trigger fires.

Getting Information About Triggers

- The execution plan for a trigger is stored in *sysprocedures*.

- Each trigger is assigned an identification number, which is stored as a new row in *sysobjects* with the object ID for the table to which it applies in the *deltrig* column, and also as an entry in the *deltrig*, *instrig*, and *updtrig* columns of the *sysobjects* row for the table to which it applies.
- To display the text of a trigger, which is stored in *syscomments*, use **sp_helptext**.

  If the System Security Officer has reset the **allow select on syscomments.text column** parameter with the system procedure **sp_configure** (as required to run Adaptive Server in the evaluated configuration), you must be the creator of the trigger or a System Administrator to view the text of the trigger through **sp_helptext**.
- For a report on a trigger, use **sp_help**.
- For a report on the tables and views that are referenced by a trigger, use **sp_depends**.

Triggers and Performance

- In performance terms, trigger overhead is usually very low. The time involved in running a trigger is spent mostly in referencing other tables, which are either in memory or on the database device.
- The *deleted* and *inserted* tables often referenced by triggers are always in memory rather than on the database device, because they are logical tables. The location of other tables referenced by the trigger determines the amount of time the operation takes.

Setting Options Within Triggers

- You can use the **set** command inside a trigger. The **set** option you invoke remains in effect during the execution of the trigger, then reverts to its former setting. In particular, the **self_recursion** option can be used inside a trigger so that data modifications by the trigger itself can cause the trigger to fire again.

Dropping a Trigger

- You must drop and re-create the trigger if you rename any of the objects referenced by the trigger. You can rename a trigger with **sp_rename**.
- When you drop a table, any triggers associated with it are also dropped.

Actions That Do Not Cause Triggers to Fire

- A **truncate table** command is not caught by a **delete** trigger. Although a **truncate table** statement is, in effect, like a **delete** without a **where** clause (it removes all rows), changes to the data rows are not logged, and so cannot fire a trigger.

  Since permission for the **truncate table** command defaults to the table owner and is not transferable, only the table owner need worry about inadvertently circumventing a **delete** trigger with a **truncate table** statement.
- The **writetext** command, whether logged or unlogged, does not cause a trigger to fire.

Triggers and Transactions

- When a trigger is defined, the action it specifies on the table to which it applies is always implicitly part of a transaction, along with the trigger itself. Triggers are often used to roll back an entire transaction if an error is detected, or they can be used roll back the effects of a specific data modification:
  - When the trigger contains the **rollback transaction** command, the rollback aborts the entire batch, and any subsequent statements in the batch are not executed.
  - When the trigger contains the **rollback trigger**, the rollback affects only the data modification that caused the trigger to fire. The **rollback trigger** command can include a **raiserror** statement. Subsequent statements in the batch are executed.
- Since triggers execute as part of a transaction, the following statements and system procedures are not allowed in a trigger:
  - All **create** commands, including **create database**, **create table**, **create index**, **create procedure**, **create default**, **create rule**, **create trigger**, and **create view**
  - All **drop** commands
  - **alter table** and **alter database**
  - **truncate table**
  - **grant** and **revoke**
  - **update statistics**
  - **sp_configure**
  - **load database** and **load transaction**

  - **disk init, disk mirror, disk refit, disk reinit, disk remirror, disk unmirror**
  - **select into**

- If a desired result (such as a summary value) depends on the number of rows affected by a data modification, use *@@rowcount* to test for multirow data modifications (an **insert**, **delete**, or **update** based on a **select** statement), and take appropriate actions. Any Transact-SQL statement that does not return rows (such as an **if** statement) sets *@@rowcount* to 0, so the test of *@@rowcount* should occur at the beginning of the trigger.

Update and Insert Triggers

- When an **insert** or **update** command executes, Adaptive Server adds rows to both the trigger table and the *inserted* table at the same time. The rows in the *inserted* table are always duplicates of one or more rows in the trigger table.
- An **update** or **insert** trigger can use the **if update** command to determine whether the **update** or **insert** changed a particular column. **if update**(*column_name*) is true for an **insert** statement whenever the column is assigned a value in the select list or in the **values** clause. An explicit NULL or a default assigns a value to a column and thus activates the trigger. An implicit NULL, however, does not.

  For example, if you create the following table and trigger:

```
create table junk
(aaa int null,
bbb int not null)

create trigger trigtest on junk
for insert as
if update (aaa)
    print "aaa updated"
if update (bbb)
    print "bbb updated"
```

  Inserting values into either column or into both columns fires the trigger for both column *aaa* and column *bbb*:

```
insert junk (aaa, bbb)
values (1, 2)

aaa updated
bbb updated
```

Inserting an explicit null into column *aaa* also fires the trigger:

```
insert junk
values (NULL, 2)

aaa updated
bbb updated
```

If there was a default for column *aaa*, the trigger would also fire.

However, with no default for column *aaa* and no value explicitly inserted, Adaptive Server generates an implicit NULL and the trigger does not fire:

```
insert junk (bbb)
values(2)

bbb updated
```

if update is never true for a **delete** statement.

Nesting Triggers and Trigger Recursion

- Adaptive Server allows nested triggers by default. To prevent triggers from nesting, use **sp_configure** to set the **allow nested triggers** option to 0 (off), as follows:

  ```
  sp_configure "allow nested triggers", 0
  ```

- Triggers can be nested to a depth of 16 levels. If a trigger changes a table on which there is another trigger, the second trigger will fire and can then call a third trigger, and so forth. If any trigger in the chain sets off an infinite loop, the nesting level will be exceeded and the trigger will abort, rolling back the transaction that contains the trigger query.

➤ ***Note***

Since triggers are put into a transaction, a failure at any level of a set of nested triggers cancels the entire transaction: all data modifications are rolled back. Supply your triggers with messages and other error handling and debugging aids in order to determine where the failure occurred.

- The global variable *@@nestlevel* contains the nesting level of the current execution. Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. If the maximum of 16 is exceeded, the transaction aborts.

- If a trigger calls a stored procedure that performs actions that would cause the trigger to fire again, the trigger is reactivated only if nested triggers are enabled. Unless there are conditions within the trigger that limit the number of recursions, this causes a nesting-level overflow.

  For example, if an update trigger calls a stored procedure that performs an update, the trigger and stored procedure execute once if **allow nested triggers** is off. If **allow nested triggers** is on, and the number of updates is not limited by a condition in the trigger or procedure, the procedure or trigger loop continues until it exceeds the 16-level maximum nesting value.
- By default, a trigger does not call itself in response to a second data modification to the same table within the trigger, regardless of the setting of the **allow nested triggers** configuration parameter. A **set** option, **self_recursion**, enables a trigger to fire again as a result of a data modification within the trigger. For example, if an update trigger on one column of a table results in an update to another column, the update trigger fires only once when **self_recursion** is disabled, but it can fire up to 16 times if **self_recursion** is set on. The **allow nested triggers** configuration parameter must also be enabled in order for self-recursion to take place.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Security Officer can grant or revoke permissions to create triggers.

Permission to issue the **create trigger** command is granted to users by default. When you revoke permission for a user to create triggers, a revoke row is added in the *sysprotects* table for that user. To grant permission to that user to issue **create trigger**, you must issue two **grant** commands. The first command removes the revoke row from *sysprotects*; the second inserts a grant row.

If you revoke permission to create triggers, the user cannot create triggers even on tables that the user owns. Revoking permission to create triggers from a user affects only the database where the revoke command was issued.

Permissions on Objects: Trigger Creation Time

When you create a trigger, Adaptive Server makes no permission checks on objects, such as tables and views, that the trigger references. Therefore, you can create a trigger successfully, even though you do not have access to its objects. All permission checks occur when the trigger fires.

Permissions on Objects: Trigger Execution Time

When the trigger executes, permission checks on its objects depend on whether the trigger and its objects are owned by the same user.

- If the trigger and its objects are not owned by the same user, the user who caused the trigger to fire must have been granted direct access to the objects. For example, if the trigger performs a select from a table the user cannot access, the trigger execution fails. In addition, the data modification that caused the trigger to fire is rolled back.
- If a trigger and its objects are owned by the same user, special rules apply. The user automatically has implicit permission to access the trigger's objects, even though the user cannot access them directly. A detailed description of the rules for implicit permissions is discussed in the *System Administration Guide*.

See Also

Commands	**alter table**, **create procedure**, **create table**, **drop trigger**, **rollback trigger**, **set**
System procedures	**sp_commonkey**, **sp_configure**, **sp_depends**, **sp_foreignkey**, **sp_help**, **sp_helptext**, **sp_primarykey**, **sp_rename**, **sp_spaceused**

create view

Function

Creates a view, which is an alternative way of looking at the data in one or more tables.

Syntax

```
create view [owner.]view_name
   [(column_name [, column_name]...)]
   as select [distinct] select_statement
   [with check option]
```

Keywords and Options

view_name – is the name of the view. The name cannot include the database name. If you have **set quoted_identifier on**, you can use a delimited identifier. Otherwise, the view name cannot be a variable and must conform to the rules for identifiers. For more information about valid view names, see "Identifiers" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters." Specify the owner's name to create another view of the same name owned by a different user in the current database. The default value for *owner* is the current user.

column_name – specifies names to be used as headings for the columns in the view. If you have set **quoted_identifier on**, you can use a delimited identifier. Otherwise, the column name must conform to the rules for identifiers. For more information about valid column names, see "Identifiers" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

It is always legal to supply column names, but column names are required only in the following cases:

- When a column is derived from an arithmetic expression, function, string concatenation, or constant
- When two or more columns have the same name (usually because of a join)
- When you want to give a column in a view a different name than the column from which it is derived (see example 3).

Column names can also be assigned in the **select** statement (see example 4). If no column names are specified, the view columns acquire the same names as the columns in the **select** statement.

select – begins the **select** statement that defines the view.

distinct – specifies that the view cannot contain duplicate rows.

select_statement – completes the **select** statement that defines the view. It can use more than one table and other views.

with check option – indicates that all data modification statements are validated against the view selection criteria. All rows inserted or updated through the view must remain visible through the view.

Examples

1. ```
   create view titles_view
   as select title, type, price, pubdate
   from titles
   ```

   Creates a view derived from the *title*, *type*, *price* and *pubdate* columns of the base table *titles*.

2. ```
   create view "new view" ("column 1", "column 2")
   as select col1, col2 from "old view"
   ```

   Creates the "new view" view from "old view." Both columns are renamed in the new view. All view and column names that include embedded blanks are enclosed in double quotation marks. Before creating the view, you must use **set quoted_identifier on**.

3. ```
   create view accounts (title, advance, amt_due)
   as select title, advance, price * total_sales
   from titles
   where price > $5
   ```

   Creates a view which contains the titles, advances and amounts due for books with a price less than $5.00.

4. ```
   create view cities
   (authorname, acity, publishername, pcity)
   as select au_lname, authors.city, pub_name,
   publishers.city
   from authors, publishers
   where authors.city = publishers.city
   ```

   Creates a view derived from two base tables, *authors* and *publishers*. The view contains the names and cities of authors who live in a city in which there is a publisher.

```
5. create view cities2
   as select authorname = au_lname,
   acity = authors.city, publishername = pub_name,
   pcity = publishers.city
   from authors, publishers
   where authors.city = publishers.city
```

Creates a view with the same definition as in example 3, but with column headings provided in the **select** statement.

```
6. create view author_codes
   as select distinct au_id
   from titleauthor
```

Creates a view, *author_codes*, derived from *titleauthor* that lists the unique author identification codes.

```
7. create view price_list (price)
   as select distinct price
   from titles
```

Creates a view, *price_list*, derived from *title* that lists the unique book prices.

```
8. create view stores_cal
   as select * from stores
   where state = "CA"
   with check option
```

Creates a view of the *stores* table that excludes information about stores outside of California. The **with check option** clause validates each inserted or updated row against the view's selection criteria. Rows for which *state* has a value other than "CA" are rejected.

```
9. create view stores_cal30
   as select * from stores_cal
   where payterms = "Net 30"
```

Creates a view, *stores_cal30*, which is derived from *stores_cal*. The new view inherits the check option from *stores_cal*. All rows inserted or updated through *stores_cal*30 must have a *state* value of "CA". Because *stores_cal30* has no **with check option** clause, it is possible to insert or update rows through *stores_cal30* for which *payterms* has a value other than "Net 30".

```
10.create view stores_cal30_check
   as select * from stores_cal
   where payterms = "Net 30"
   with check option
```

Creates a view, *stores_cal30_check*, derived from *stores_cal*. The new view inherits the check option from *stores_cal*. It also has a **with check option** clause of its own. Each row that is inserted or updated through *stores_cal30_check* is validated against the selection criteria of both *stores_cal* and *stores_cal30_check*. Rows with a *state* value other than "CA" or a *payterms* value other than "Net 30" are rejected.

Comments

- You can use views as security mechanisms by granting permission on a view, but not on its underlying tables.
- You can rename a view with **sp_rename**.
- When you query through a view, Adaptive Server checks to make sure that all the database objects referenced anywhere in the statement exist, that they are valid in the context of the statement, and that data update commands do not violate data integrity rules. If any of these checks fail, you get an error message. If the checks are successful, **create view** "translates" the view into an action on the underlying table(s).
- For more information about views, see the *Transact-SQL User's Guide*.

Restrictions on Views

- You can create a view only in the current database.
- The number of columns referenced by a view cannot exceed 250.
- You cannot create a view on a temporary table.
- You cannot create a trigger or build an index on a view.
- You cannot use **readtext** or **writetext** on *text* or *image* columns in views.
- You cannot include **order by** or **compute** clauses, the keyword **into**, or the **union** operator in the **select** statements that define views.
- **create view** statements can be combined with other SQL statements in a single batch.

◆ ***WARNING!***

When a create view command occurs within an if...else block or a while loop, Adaptive Server creates the schema for the view before determining whether the condition is true. This may lead to errors if the view already exists. Make sure a view with the same name does not already exist in the database.

View Resolution

- If you alter the structure of a view's underlying table(s) by adding or deleting columns, the new columns will not appear in a view defined with a **select** * clause unless the view is dropped and redefined. The asterisk shorthand is interpreted and expanded when the view is first created.
- If a view depends on a table (or view) that has been dropped, Adaptive Server produces an error message when anyone tries to use the view. If a new table (or view) with the same name and schema is created to replace the one that has been dropped, the view again becomes usable.
- You can redefine a view without redefining other views that depend on it, unless the redefinition makes it impossible for Adaptive Server to translate the dependent view(s).

Modifying Data Through Views

- **delete** statements are not allowed on multitable views.
- **insert** statements are not allowed unless all **not null** columns in the underlying table or view are included in the view through which you are inserting new rows. (Adaptive Server cannot supply values for **not null** columns in the underlying table or view.)
- You cannot insert a row through a view that includes a computed column.
- **insert** statements are not allowed on join views created with **distinct** or **with check option.**
- **update** statements are allowed on join views **with check option**. The update fails if any of the affected columns appears in the **where** clause, in an expression that includes columns from more than one table.
- If you insert or update a row through a join view, all affected columns must belong to the same base table.

- You cannot update or insert into a view defined with the **distinct** clause.
- Data update statements cannot change any column in a view that is a computation and cannot change a view that includes aggregates.

IDENTITY Columns and Views

- You cannot add a new IDENTITY column to a view with the *column_name* = **identity**(*precision*) syntax.
- To insert an explicit value into an IDENTITY column, the table owner, Database Owner, or System Administrator must **set identity_insert** *table_name* **on** for the column's base table, not through the view through which it is being inserted.

group by Clauses and Views

- When creating a view for security reasons, be careful when using aggregate functions and the **group by** clause. A Transact-SQL extension allows you to name columns that do not appear in the **group by** clause. If you name a column that is not in the **group by** clause, Adaptive Server returns detailed data rows for the column. For example, this query:

  ```
  select title_id, type, sum(total_sales)
  from titles
  group by type
  ```

  returns a row for every (18 rows)—more data than you might intend. While this query:

  ```
  select type, sum(total_sales)
  from titles
  group by type
  ```

  returns one row for each type (6 rows).

  For more information about **group by**, see "group by and having Clauses."

distinct Clauses and Views

- The **distinct** clause defines a view as a database object that contains no duplicate rows. A row is defined to be a duplicate of another row if all of its column values match the same column values in another row. Null values are considered to be duplicates of other null values.

Querying a subset of a view's columns can result in what appear to be duplicate rows. If you select a subset of columns, some of which contain the same values, the results appear to contain duplicate rows. However, the underlying rows in the view are still unique. Adaptive Server applies the **distinct** requirement to the view's definition when it accesses the view for the first time (before it does any projection and selection) so that all the view's rows are distinct from each other.

You can specify **distinct** more than once in the view definition's **select** statement to eliminate duplicate rows, as part of an aggregate function or a **group by** clause. For example:

```
select distinct count(distinct title_id), price
from titles
```

- The scope of the **distinct** applies only for that view; it does not cover any new views derived from the **distinct** view.

with check option Clauses and Views

- If a view is created **with check option**, each row that is inserted or updated through the view must meet the selection criteria of the view.
- If a view is created **with check option**, all views derived from the "base" view must satisfy its check option. Each row inserted or updated through the derived view must remain visible through the base view.

Getting Information About Views

- To get a report of the tables or views on which a view depends, and of objects that depend on a view, execute the system procedure **sp_depends**.
- To display the text of a view, which is stored in *syscomments*, execute the system procedure **sp_helptext** with the view name as the parameter.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of more than one **distinct** keyword and the use of "*column_heading* = *column_name*" in the **select** list are Transact-SQL extensions.

Permissions

create view permission defaults to the Database Owner, who can transfer it to other users.

Permissions on Objects: View Creation Time

When you create a view, Adaptive Server makes no permission checks on objects, such as tables and views, that are referenced by the view. Therefore, you can create a view successfully even if you do not have access to its objects. All permission checks occur when a user invokes the view.

Permissions on Objects: View Execution Time

When a view is invoked, permission checks on its objects depend on whether the view and all referenced objects are owned by the same user.

- If the view and its objects are not owned by the same user, the invoker must have been granted direct access to the objects. For example, if the view performs a select from a table the invoker cannot access, the select statement fails.
- If the view and its objects are owned by the same user, special rules apply. The invoker automatically has implicit permission to access the view's objects even though the invoker could not access them directly. Without having to grant users direct access to your tables, you can give them restricted access with a view. In this way, a view can be a security mechanism. For example, invokers of the view might be able to access only certain rows and columns of your table. A detailed description of the rules for implicit permissions is discussed in the *System Administration Guide*.

See Also

Commands	**create schema**, **drop view**, **update**
System procedures	**sp_depends**, **sp_help**, **sp_helptext**, **sp_rename**

dbcc

Function

Database Consistency Checker (**dbcc**) checks the logical and physical consistency of a database and provides statistics, planning, and repair functionality.

Syntax

```
dbcc checkalloc [(database_name [, fix | nofix])]
dbcc checkcatalog [(database_name)]
dbcc checkdb [(database_name [, skip_ncindex])]
dbcc checkstorage [(database_name)]
dbcc checktable({table_name|table_id}[, skip_ncindex])
dbcc checkverify [(database_name)]
dbcc complete_xact (xid, {"commit" | "rollback"})
dbcc forget_xact (xid)
dbcc dbrepair (database_name, dropdb)
dbcc engine( {offline , [enginenum] | "online" })
dbcc fix_text ({table_name | table_id})
dbcc indexalloc ({table_name | table_id}, index_id
      [, {full | optimized | fast | null}
      [, fix | nofix]])
dbcc rebuild_text (table [, column
   [, text_page_number]])
dbcc reindex ({table_name | table_id})
dbcc tablealloc ({table_name | table_id}
      [, {full | optimized | fast | null}
      [, fix | nofix]])|
dbcc { traceon | traceoff } (flag [, flag ... ])
dbcc tune ( { ascinserts, {0 | 1 } , tablename |
              cleanup, {0 | 1 } |
              cpuaffinity, start_cpu {, on| off } |
              des_greedyalloc, dbid, object_name,
                  " { on|off }" |
              deviochar vdevno, "batch_size" |
              doneinproc { 0 | 1 } |
              maxwritedes, writes_per_batch } )
```

Keywords and Options

checkalloc – checks the specified database to see that all pages are correctly allocated and that no page that is allocated is not used. If no database name is given, **checkalloc** checks the current database. It always uses the **optimized** report option (see **tablealloc**).

checkalloc reports on the amount of space allocated and used.

database_name – is the name of the database to check. If no database name is given, **dbcc** uses the current database.

fix | **nofix** – determines whether **dbcc** fixes the allocation errors found. The default mode for **checkalloc** is **nofix**. You must put the database into single-user mode in order to use the **fix** option.

For a discussion of page allocation in Adaptive Server, see the *System Administration Guide*.

checkcatalog – checks for consistency in and between system tables. For example, it makes sure that every type in *syscolumns* has a matching entry in *systypes*, that every table and view in *sysobjects* has at least one column in *syscolumns*, and that the last checkpoint in *syslogs* is valid. **checkcatalog** also reports on any segments that have been defined. If no database name is given, **checkcatalog** checks the current database.

checkdb – runs the same checks as **checktable**, but on each table, including *syslogs*, in the specified database. If no database name is given, **checkdb** checks the current database.

skip_ncindex – causes **dbcc checktable** or **dbcc checkdb** to skip checking the nonclustered indexes on user tables. The default is to check all indexes.

checkstorage – checks the specified database for allocation, OAM page entries, page consistency, text valued columns, allocation of text valued columns, and text column chains. The results of each **dbcc checkstorage** operation are stored in the *dbccdb* database. For details on using **dbcc checkstorage**, and on creating, maintaining, and generating reports from *dbccdb*, see the *System Administration Guide*.

checktable – checks the specified table to see that index and data pages are correctly linked, that indexes are in properly sorted order, that all pointers are consistent, that the data information on each page is reasonable, and that page offsets are reasonable. If the log

segment is on its own device, running **dbcc checktable** on the *syslogs* table reports the log(s) used and free space. For example:

```
Checking syslogs
The total number of data pages in this table is 1.
*** NOTICE: Space used on the log segment is 0.20 Mbytes, 0.13%.
*** NOTICE: Space free on the log segment is 153.4 Mbytes, 99.87%.
DBCC execution completed.  If dbcc printed error messages, see your
System Administrator.
```

If the log segment is not on its own device, the following message appears:

```
*** NOTICE:  Notification of log space used/free cannot be
reported because the log segment is not on its own device.
```

table_name | *table_id* – is the name or object ID of the table to check.

checkverify – verifies the results of the most recent run of **dbcc checkstorage** for the specified database. For details on using **dbcc checkverify**, see the *System Administration Guide*.

complete_xact – heuristically completes a transaction by either committing or rolling back its work. Adaptive Server retains information about all heuristically completed transactions in the *master.dbo.systransactions* table, so that the external transaction coordinator may have some knowledge of how the transaction was completed.

◆ ***WARNING!***

Heuristically completing a transaction in the prepared state can cause inconsistent results for an entire distributed transaction. The System Administrator's decision to heuristically commit or roll back a transaction may contradict the decision made by the coordinating Adaptive Server or protocol.

forget_xact – removes the commit status of a heuristically completed transaction from *master.dbo.systransactions*. **forget_xact** can be used when the System Administrator does not want the coordinating service to have knowledge that a transaction was heuristically completed, or when an external coordinator will not be available to clear commit status in *systransactions*.

◆ ***WARNING!***

Never use *dbcc forget_xact* in a normal DTP environment, since the external transaction coordinator should be permitted to detect heuristically-completed transactions. X/Open XA-compliant transaction managers and Adaptive Server transaction coordination services automatically clear the commit status in *systransactions*.

xid – is a transaction name from the *systransactions.xactname* column. You can also determine valid *xid* values using **sp_transactions**.

dbrepair (*database_name*, **dropdb**) – drops a damaged database. The **drop database** command does not work on a damaged database.

Users cannot be using the database being dropped when this **dbcc** statement is issued (including the user issuing the statement).

fengine – takes Adaptive Server engines offline or brings them online. If *enginenum* is not specified, **dbcc engine (offline)** takes the highest-numbered engine offline. For more information, see Chapter 16, "Managing Multiprocessor Servers," in the *System Administration Guide*.

fix_text – upgrades *text* values after an Adaptive Server's character set has been changed from any character set to a new multibyte character set.

Changing to a multibyte character set makes the internal management of *text* data more complicated. Since a *text* value can be large enough to cover several pages, Adaptive Server must be able to handle characters that span page boundaries. To do so, the server requires additional information on each of the *text* pages. The System Administrator or table owner must run **dbcc fix_text** on each table that has *text* data to calculate the new values needed. For more information, see the *System Administration Guide*.

indexalloc – checks the specified index to see that all pages are correctly allocated and that no page that is allocated is not used. This is a smaller version of **checkalloc**, providing the same integrity checks on an individual index.

indexalloc produces the same three types of reports as **tablealloc**: **full**, **optimized**, and **fast**. If no type is indicated, or if you use **null**, Adaptive Server uses **optimized**. The **fix|nofix** option functions the same with **indexalloc** as with **tablealloc**.

➤ ***Note***

You can specify **fix** or **nofix** only if you include a value for the type of report (**full**, **optimized**, **fast**, or **null**).

table_name | *table_id, index_id* – is the table name or the table's object ID (the *id* column from *sysobjects)* plus the index's *indid* from *sysindexes*.

full – reports all types of allocation errors.

optimized – produces a report based on the allocation pages listed in the object allocation map (OAM) pages for the index. It does not report and cannot fix unreferenced extents on allocation pages that are not listed in the OAM pages. The **optimized** option is the default.

fast – does not produce an allocation report, but produces an exception report of pages that are referenced but not allocated in the extent (2521-level errors).

fix | **nofix** – determines whether **indexalloc** fixes the allocation errors found in the table. The default is **fix** for all indexes except indexes on system tables, for which the default is **nofix**. To use the **fix** option with system tables, you must first put the database in single-user mode.

You can specify **fix** or **nofix** only if you include a value for the type of report (**full**, **optimized**, **fast**, or **null**).

rebuild_text – rebuilds or creates an internal Adaptive Server 12.x data structure for *text* or *image* data. This data structure enables Adaptive Server to perform random access and asynchronous prefetch during data queries.

reindex – checks the integrity of indexes on user tables by running a fast version of **dbcc checktable**. It can be used with the table name or the table's object ID (the *id* column from *sysobjects)*. **reindex** prints a message when it discovers the first index-related error, then drops and re-creates the suspect indexes. The System Administrator or table owner must run **dbcc reindex** after Adaptive

Server's sort order has been changed and indexes have been marked "suspect" by Adaptive Server.

When **dbcc** finds corrupt indexes, it drops and re-creates the appropriate indexes. If the indexes for a table are already correct, or if the table has no indexes, **dbcc reindex** does not rebuild the index, but prints an informational message instead.

dbcc reindex aborts if a table is suspected of containing corrupt data. When that happens, an error message instructs the user to run **dbcc checktable**. **dbcc reindex** does not allow reindexing of system tables. System indexes are checked and rebuilt, if necessary, as an automatic part of recovery after Adaptive Server is restarted following a sort order change.

tablealloc – checks the specified table to see that all pages are correctly allocated and that no page that is allocated is not used. This is a smaller version of **checkalloc**, providing the same integrity checks on an individual table. It can be used with the table name or the table's object ID (the *id* column from *sysobjects).* For an example of **tablealloc** output, see the *System Administration Guide.*

Three types of reports can be generated with **tablealloc**: **full**, **optimized**, and **fast.** If no type is indicated, or if you use **null**, Adaptive Server uses **optimized.**

full – is equivalent to **checkalloc** at a table level; it reports all types of allocation errors.

optimized – produces a report based on the allocation pages listed in the object allocation map (OAM) pages for the table. It does not report and cannot fix unreferenced extents on allocation pages that are not listed in the OAM pages. The **optimized** option is the default.

fast – does not produce an allocation report, but produces an exception report of pages that are referenced but not allocated in the extent (2521-level errors).

fix | **nofix** – determines whether or not **tablealloc** fixes the allocation errors found in the table. The default is **fix** for all tables except system tables, for which the default is **nofix**. To use the **fix** option with system tables, you must first put the database in single user mode.

You can specify **fix** or **nofix** only if you include a value for the type of report (**full**, **optimized**, **fast**, or **null**).

traceon | traceoff – toggles the printing of diagnostics during query optimization (*flag* values 302, 310, and 317). Values 3604 and 3605 toggle sending trace output to the user session and to the error log, respectively. For more information, see "Tuning with dbcc traceon" in the *Performance and Tuning Guide*.

tune – enables or disables tuning flags for special performance situations. For more information on the individual options, see the *Performance and Tuning Guide*.

Examples

1. dbcc checkalloc(pubs2)

Checks *pubs2* for page allocation errors.

2. dbcc checkstorage(pubs2)

Checks database consistency for *pubs2* and places the information in the *dbccdb* database.

3. dbcc tablealloc(publishers, null, nofix)

Adaptive Server returns an optimized report of allocation for this table, but does not fix any allocation errors.

4. dbcc checktable(salesdetail)

```
Checking salesdetail
The total number of pages in partition 1 is 3.
The total number of pages in partition 2 is 1.
The total number of pages in partition 3 is 1.
The total number of pages in partition 4 is 1.
The total number of data pages in this table is 10.
Table has 116 data rows.
DBCC execution completed. If DBCC printed error messages,
contact a user with System Administrator (SA) role.
```

5. dbcc indexalloc ("pubs..titleauthor", 2, full)

Adaptive Server returns a full report of allocation for the index with an *indid* of 2 on the *titleauthor* table and fixes any allocation errors.

6. dbcc rebuild_text (blurbs)

Rebuilds or creates an internal Adaptive Server 12.x data structure for all *text* and *image* columns in the *blurbs* table.

7. dbcc reindex(titles)

```
One or more indexes are corrupt. They will be
     rebuilt.
```

dbcc reindex has discovered one or more corrupt indexes in the *titles* table.

8. `dbcc fix_text(blurbs)`

   Upgrades text values for *blurbs* after a character set change.

9. `dbcc complete_xact (distributedxact1, "rollback")`

   Heuristically aborts the transaction, "distributedxact1."

10. `dbcc forget_xact (distributedxact1)`

    Removes information for the transaction, "distributedxact1" from *master.dbo.systransactions*.

Comments

- **dbcc**, the Database Consistency Checker, can be run while the database is active, except for the **dbrepair(*database_name*, dropdb)** option and **dbcc checkalloc** with the **fix** option.
- **dbcc** locks database objects as it checks them. For information on minimizing performance problems while using **dbcc**, see the **dbcc** discussion in the *System Administration Guide*.
- To qualify a table or an index name with a user name or database name, enclose the qualified name in single or double quotation marks. For example:

  ```
  dbcc tablealloc("pubs2.pogo.testtable")
  ```

- **dbcc reindex** cannot be run within a user-defined transaction.
- **dbcc fix_text** can generate a large number of log records, which may fill up the transaction log. **dbcc fix_text** is designed so that updates are done in a series of small transactions: in case of a log space failure, only a small amount of work is lost. If you run out of log space, clear your log and restart **dbcc fix_text** using the same table that was being upgraded when the original **dbcc fix_text** failed.
- If you attempt to use **select**, **readtext**, or **writetext** on *text* values after changing to a multibyte character set, and you have not run **dbcc fix_text**, the command fails, and an error message instructs you to run **dbcc fix_text** on the table. However, you can delete *text* rows after changing character sets without running **dbcc fix_text**.
- **dbcc** output is sent as messages or errors, rather than as result rows. Client programs and scripts should check the appropriate error handlers.
- If a table is partitioned, **dbcc checktable** returns information about each partition.

- *text* and *image* data that has been upgraded to Adaptive Server version 12.x **is not** automatically upgraded to the new storage format. To improve query performance and enable prefetch for this data, use the **rebuild_text** keyword against the upgraded *text* and *image* columns.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the table owner can execute **dbcc** with the **checktable**, **fix_text**, **rebuild_text**, or **reindex** keywords. Only the Database Owner can use the **checkstorage**, **checkdb**, **checkcatalog**, **checkalloc**, **indexalloc**, and **tablealloc** keywords. Only a System Administrator can use the **dbrepair**, **complete_xact**, and **forget_xact** keywords. Only a System Administrator can use **dbcc traceon** and **dbcc traceoff** commands. Only a System Administrator can use **dbcc engine**.

See Also

Commands	drop database
System procedures	sp_configure, sp_helpdb

deallocate cursor

Function

Makes a cursor inaccessible and releases all memory resources committed to that cursor.

Syntax

```
deallocate cursor cursor_name
```

Parameters

cursor_name – is the name of the cursor to deallocate.

Examples

```
1. deallocate cursor authors_crsr
```

Deallocates the cursor named "authors_crsr."

Comments

- Adaptive Server returns an error message if the cursor does not exist.
- You must deallocate a cursor before you can use its cursor name as part of another **declare cursor** statement.
- **deallocate cursor** has no effect on memory resource usage when specified in a stored procedure or trigger.
- You can **deallocate** a cursor whether it is open or closed.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

deallocate cursor permission defaults to all users. No permission is required to use it.

See Also

Commands	close, declare cursor

declare

Function

Declares the name and type of local variables for a batch or procedure.

Syntax

Variable declaration:

```
declare @variable_name datatype
   [, @variable_name datatype]...
```

Variable assignment:

```
select @variable = {expression | select_statement}
   [, @variable = {expression | select_statement} ...]
   [from table_list]
   [where search_conditions]
   [group by group_by_list]
   [having search_conditions]
   [order by order_by_list]
   [compute function_list [by by_list]]
```

Keywords and Options

@variable_name – must begin with @ and must conform to the rules for identifiers.

datatype – can be either a system datatype or a user-defined datatype.

Examples

```
1. declare @one varchar(18), @two varchar(18)
   select @one = "this is one", @two = "this is two"
   if @one = "this is one"
     print "you got one"
   if @two = "this is two"
     print "you got two"
   else print "nope"
```

```
you got one
you got two
```

Declares two variables and prints strings according to the values in the variables.

```
2. declare @veryhigh money
   select @veryhigh = max(price)
     from titles
   if @veryhigh > $20
     print "Ouch!"
```

Prints “Ouch!” if the maximum book price in the *titles* table is more than $20.00.

Comments

- Assign values to local variables with a **select** statement.
- The maximum number of parameters in a procedure is 255. The number of local or global variables is limited only by available memory. The @ sign denotes a variable name.
- Local variables are often used as counters for **while** loops or **if...else** blocks. In stored procedures, they are declared for automatic, noninteractive use by the procedure when it executes. Local variables must be used in the batch or procedure in which they are declared.
- The **select** statement that assigns a value to the local variable usually returns a single value. If there is more than one value to return, the variable is assigned the last one. The **select** statement that assigns values to variables cannot be used to retrieve data in the same statement.
- The **print** and **raiserror** commands can take local variables as arguments.
- Users cannot create global variables and cannot update the value of global variables directly in a **select** statement.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

declare permission defaults to all users. No permission is required to use it.

See Also

Commands	**print**, **raiserror**, **select**, **while**

declare cursor

Function

Defines a cursor.

Syntax

```
declare cursor_name cursor
   for select_statement
   [for {read only | update [of column_name_list]}]
```

Parameters

cursor_name – is the name of the cursor being defined.

select_statement – is the query that defines the cursor result set. See **select** for more information.

for read only – specifies that the cursor result set cannot be updated.

for update – specifies that the cursor result set is updatable.

of *column_name_list* – is the list of columns from the cursor result set (specified by the *select_statement*) defined as updatable. Adaptive Server also allows you to include columns that are not specified in the list of columns of the cursor's *select_statement* (and excluded from the result set), but that are part of the tables specified in the *select_statement*.

Examples

1.
```
declare authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state != 'CA'
```

   Defines a result set for the *authors_crsr* cursor that contains all authors from the *authors* table who do not reside in California.

2.
```
declare titles_crsr cursor
for select title, title_id from titles
where title_id like "BU%"
for read only
```

   Defines a read-only result set for the *titles_crsr* cursor that contains the business-type books from the *titles* table.

```
3. declare pubs_crsr cursor
   for select pub_name, city, state
   from publishers
   for update of city, state
```

Defines an updatable result set for the *pubs_crsr* cursor that contains all of the rows from the *publishers* table. It defines the address of each publisher (*city* and *state* columns) for update.

Comments

Restrictions on Cursors

- A **declare cursor** statement must precede any **open** statement for that cursor.
- You cannot include other statements with **declare cursor** in the same Transact-SQL batch.
- *cursor_name* must be a valid Adaptive Server identifier.

Cursor *select* Statements

- *select_statement* can use the full syntax and semantics of a Transact-SQL **select** statement, with these restrictions:
  - *select_statement* must contain a **from** clause.
  - *select_statement* cannot contain a **compute**, **for browse**, or **into** clause.
  - *select_statement* can contain the **holdlock** keyword.
- The *select_statement* can contain references to Transact-SQL parameter names or Transact-SQL local variables (for all cursor types except language). The names must reference the Transact-SQL parameters and local variables defined in the procedure, trigger, or statement batch that contains the **declare cursor** statement.

  The parameters and local variables referenced in the **declare cursor** statement do not have to contain valid values until the cursor is opened.
- The *select_statement* can contain references to the *inserted* and *deleted* temporary tables that are used in triggers.

Scope

- A cursor's existence depends on its **scope**. The scope refers to the context in which the cursor is used, that is, within a user session, within a stored procedure, or within a trigger.

Within a user session, the cursor exists only until the user ends the session. The cursor does not exist for any additional sessions started by other users. After the user logs off, Adaptive Server deallocates the cursors created in that session.

If a **declare cursor** statement is part of a stored procedure or trigger, the cursor created within it applies to stored procedure or trigger scope and to the scope that launched the stored procedure or trigger. Cursors declared inside a trigger on an *inserted* or a *deleted* table are not accessible to any nested stored procedures or triggers. However, cursors declared inside a trigger on an *inserted* or a *deleted* table **are** accessible within the scope of the trigger. Once the stored procedure or trigger completes, Adaptive Server deallocates the cursors created within it.

Figure 6-1 illustrates how cursors operate between scopes.

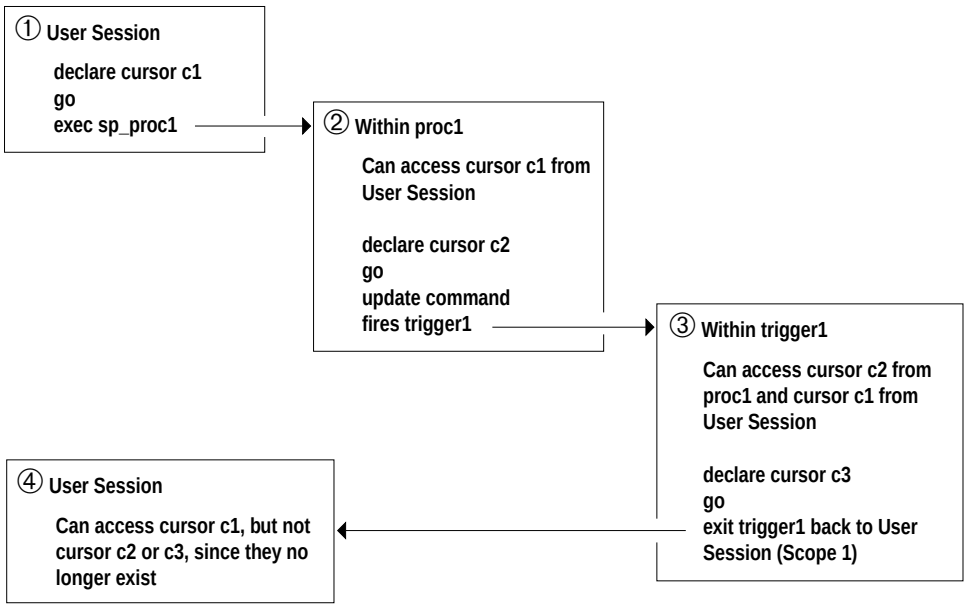

Figure 6-1: How cursors operate within scopes

- A cursor name must be unique within a given scope. Adaptive Server detects name conflicts within a particular scope only during run time. A stored procedure or trigger can define two cursors with the same name if only one is executed. For example,

the following stored procedure works because only one *names_crsr* cursor is defined in its scope:

```
create procedure proc2 @flag int
as
if @flag > 0
    declare names_crsr cursor
    for select au_fname from authors
else
    declare names_crsr cursor
    for select au_lname from authors
return
```

Result Set

- Cursor result set rows may not reflect the values in the actual base table rows. For example, a cursor declared with an **order by** clause usually requires the creation of an internal table to order the rows for the cursor result set. Adaptive Server does not lock the rows in the base table that correspond to the rows in the internal table, which permits other clients to update these base table rows. In that case, the rows returned to the client from the cursor result set would not be in sync with the base table rows.
- A cursor result set is generated as the rows are returned through a **fetch** of that cursor. This means that a cursor **select** query is processed like a normal **select** query. This process, known as a **cursor scan**, provides a faster turnaround time and eliminates the need to read rows that are not required by the application.

  A restriction of cursor scans is that they can only use the unique indexes of a table. However, if none of the base tables referenced by the cursor result set are updated by another process in the same lock space as the cursor, the restriction is unnecessary. Adaptive Server allows the declaration of cursors on tables without unique indexes, but any attempt to update those tables in the same lock space closes all cursors on the tables.

Updatable Cursors

- After defining a cursor using **declare cursor**, Adaptive Server determines whether the cursor is **updatable** or **read-only**. If a cursor is updatable, you can update or delete rows within the cursor result set. If a cursor is read-only, you cannot change the result set.

- Use the **for update** or **for read only** clause to explicitly define a cursor as updatable or read-only. You cannot define an updatable cursor if its *select_statement* contains one of the following constructs:
  - **distinct** option
  - **group by** clause
  - Aggregate function
  - Subquery
  - **union** operator
  - **at isolation read uncommitted** clause

  If you omit either the **for update** or the **read only** clause, Adaptive Server checks to see whether the cursor is updatable.

  Adaptive Server also defines a cursor as read-only if you declare a language- or server-type cursor that includes an **order by** clause as part of its *select_statement*. Adaptive Server handles updates differently for client- and execute-type cursors, thereby eliminating this restriction.
- If you do not specify a *column_name_list* with the **for update** clause, all the specified columns in the query are updatable. Adaptive Server attempts to use unique indexes for updatable cursors when scanning the base table. For cursors, Adaptive Server considers an index containing an IDENTITY column to be unique, even if it is not so declared.

  If you do not specify the **for update** clause, Adaptive Server chooses any unique index, although it can also use other indexes or table scans if no unique index exists for the specified table columns. However, when you specify the **for update** clause, Adaptive Server must use a unique index defined for one or more of the columns to scan the base table. If none exists, it returns an error.
- In most cases, include only columns to be updated in the *column_name_list* of the **for update** clause. If the table has only one unique index, you do not need to include its column in the **for update** *column_name_list*; Adaptive Server will find it when it performs the cursor scan. If the table has more than one unique index, include its column in the **for update** *column_name_list*, so that Adaptive Server can find it quickly for the cursor scan.

This allows Adaptive Server to use that unique index for its cursor scan, which helps prevent an update anomaly called the **Halloween problem**. Another way to prevent the Halloween problem is to create tables with the **unique auto_identity index** database option. For more information, see the *System Administration Guide.*

This problem occurs when a client updates a column of a cursor result set row that defines the order in which the rows are returned from the base tables. For example, if Adaptive Server accesses a base table using an index, and the index key is updated by the client, the updated index row can move within the index and be read again by the cursor. This is a result of an updatable cursor only logically creating a cursor result set. The cursor result set is actually the base tables that derive the cursor.

- If you specify the **read only** option, the cursor result set cannot be updated using the **delete** or **update** statement.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The **for update** and **for read only** options are Transact-SQL extensions.

Permissions

declare cursor permission defaults to all users. No permission is required to use it.

See Also

Commands	open

delete

Function

Removes rows from a table; the **readpast** option allows the **delete** command to skip locked rows without blocking.

Syntax

```
delete [from]
   [[database.]owner.]{view_name|table_name}
   [where search_conditions]
   [plan "abstract plan"]

delete [[database.]owner.]{table_name | view_name}
   [from [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}
    [, [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]} ...]
   [where search_conditions] ]
   [plan "abstract plan"]

delete [from]
   [[database.]owner.]{table_name|view_name}
   where current of cursor_name
```

Keywords and Options

from – (after **delete**) is an optional keyword used for compatibility with other versions of SQL.

view_name | *table_name* – is the name of the view or table from which to remove rows. Specify the database name if the view or table is in another database, and specify the owner's name if more than one view or table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

where – is a standard **where** clause. See "where Clause" for more information.

from – (after *table_name* or *view_name*) lets you name more than one table or view to use with a **where** clause when specifying which rows to delete. This **from** clause allows you to delete rows from

one table based on data stored in other tables, giving you much of the power of an embedded **select** statement.

readpast – specifies that the **delete** command skip all pages or rows on which incompatible locks are held, without waiting for locks or timing out. For datapages-locked tables, the command skips all rows on pages on which incompatible locks are held; for datarows-locked tables, it skips all rows on which incompatible locks are held.

index *index_name* – specifies an index to use for accessing *table_name*. You cannot use this option when you delete from a view.

prefetch *size* – specifies the I/O size, in kilobytes, for tables that are bound to caches with large I/Os configured. Valid values for size are 2, 4, 8, and 16. You cannot use this option when you delete from a view. **sp_helpcache** shows the valid sizes for the cache an object is bound to or for the default cache.

If Component Integration Services is enabled, you cannot use the **prefetch** keyword for remote servers.

lru | mru – specifies the buffer replacement strategy to use for the table. Use **lru** to force the optimizer to read the table into the cache on the MRU/LRU (most recently used/least recently used) chain. Use **mru** to discard the buffer from cache, and replace it with the next buffer for the table. You cannot use this option when you delete from a view.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

where current of *cursor_name* – causes Adaptive Server to delete the row of the table or view indicated by the current cursor position for *cursor_name*.

Examples

1. ```
   delete authors
   ```

   Deletes all rows from the *authors* table.

2. ```
   delete from authors
   where au_lname = "McBadden"
   ```

   Deletes a row or rows from the *authors* table.

3.
```
delete titles
from titles, authors, titleauthor
where authors.au_lname = 'Bennet'
  and authors.au_id = titleauthor.au_id
  and titleauthor.title_id = titles.title_id
```

   Deletes rows for books written by Bennet from the *titles* table. (The *pubs2* database includes a trigger (*deltitle*) that prevents the deletion of the titles recorded in the *sales* table; drop this trigger for this example to work.)

4.
```
delete titles where current of title_crsr
```

   Deletes a row from the *titles* table currently indicated by the cursor *title_crsr*.

5.
```
delete authors
where syb_identity = 4
```

   Determines which row has a value of 4 for the IDENTITY column and deletes it from the *authors* table. Note the use of the **syb_identity** keyword instead of the actual name of the IDENTITY column.

6.
```
delete from authors from authors readpast
where state = "CA"
```

   Deletes rows from *authors*, skipping any locked rows.

7.
```
delete stores from stores readpast, authors
where stores.city = authors.city
```

   Deletes rows from *stores*, skipping any locked rows. If any rows in *authors* are locked, the query blocks on these rows, waiting for the locks to be released.

Comments

- **delete** removes rows from the specified table.
- You can refer to up to 15 tables in a **delete** statement.

Restrictions

- You cannot use **delete** with a multitable view (one whose **from** clause names more than one table), even though you may be able to use **update** or **insert** on that same view. Deleting a row through a multitable view would change multiple tables, which is not permitted. **insert** and **update** statements that affect only one base table of the view are permitted.

- Adaptive Server treats two different designations for the same table in a **delete** as two tables. For example, the following **delete** issued in *pubs2* specifies *discounts* as two tables (*discounts* and *pubs2..discounts)*:

  ```
  delete discounts
  from pubs2..discounts, pubs2..stores
  where pubs2..discounts.stor_id =
      pubs2..stores.stor_id
  ```

  In this case, the join does not include *discounts*, so the **where** condition remains true for every row; Adaptive Server deletes all rows in *discounts* (which is not the desired result). To avoid this problem, use the same designation for a table throughout the statement.

- If you are deleting a row from a table that is referenced from other tables via referential constraints, Adaptive Server checks all the referencing tables before permitting the delete. If the row you are attempting to delete contains a primary key that is being used as a foreign key by one of the referencing tables, the delete is not allowed.

Deleting All Rows from a Table

- If you do not use a **where** clause, **all** rows in the table named after **delete** [**from**] are removed. The table, though empty of data, continues to exist until you issue a **drop table** command.
- **truncate table** and **delete** without a row specification are functionally equivalent, but **truncate table** is faster. **delete** removes rows one at a time and logs these transactions. **truncate table** removes whole data pages, and the rows are not logged.

  Both **delete** and **truncate table** reclaim the space occupied by the data and its associated indexes.
- You cannot use the **truncate table** command on a partitioned table. To remove all rows from a partitioned table, either use the **delete** command without a **where** clause or unpartition the table before issuing the **truncate table** command.

delete and Transactions

- In chained transaction mode, each **delete** statement implicitly begins a new transaction if no transaction is currently active. Use **commit** to complete any deletes, or use **rollback** to undo the changes. For example:

```
delete from sales where date < '01/01/89'
if exists (select stor_id
    from stores
    where stor_id not in
    (select stor_id from sales))
        rollback transaction
else
        commit transaction
```

This batch begins a transaction (using the chained transaction mode) and deletes rows with dates earlier than Jan. 1, 1989 from the *sales* table. If it deletes all sales entries associated with a store, it rolls back all the changes to *sales* and ends the transaction. Otherwise, it commits the deletions and ends the transaction. For more information about the chained mode, see the *Transact-SQL User's Guide.*

Delete Triggers

- You can define a trigger that will take a specified action when a **delete** command is issued on a specified table.

Using *delete where current of*

- Use the clause **where current of** with cursors. Before deleting rows using the clause **where current of**, you must first define the cursor with **declare cursor** and open it using the **open** statement. Position the cursor on the row you want to delete using one or more **fetch** statements. The cursor name cannot be a Transact-SQL parameter or local variable. The cursor must be an updatable cursor or Adaptive Server returns an error. Any deletion to the cursor result set also affects the base table row from which the cursor row is derived. You can delete only one row at a time using the cursor.
- You cannot delete rows in a cursor result set if the cursor's **select** statement contains a join clause, even though the cursor is considered updatable. The *table_name* or *view_name* specified with a **delete...where current of** must be the table or view specified in the first **from** clause of the **select** statement that defines the cursor.
- After the deletion of a row from the cursor's result set, the cursor is positioned before the next row in the cursor's result set. You must issue a **fetch** to access the next row. If the deleted row is the last row of the cursor result set, the cursor is positioned after the last row of the result set. The following describes the position and behavior of open cursors affected by a **delete**:

- If a client deletes a row (using another cursor or a regular **delete**) and that row represents the current cursor position of other opened cursors owned by the same client, the position of each affected cursor is implicitly set to precede the next available row. However, it is not possible for one client to delete a row representing the current cursor position of another client's cursor.
- If a client deletes a row that represents the current cursor position of another cursor defined by a join operation and owned by the same client, Adaptive Server accepts the **delete** statement. However, it implicitly closes the cursor defined by the join.

Using *readpast*

- The **readpast** option allows **delete** commands on data-only-locked tables to proceed without being blocked by incompatible locks held by other tasks.
  - On datarows-locked tables, **readpast** skips all rows on which shared, update, or exclusive locks are held by another task.
  - On datapages-locked tables, **readpast** skips all pages on which shared, update, or exclusive locks are held by another task.
- Commands specifying **readpast** block if there is an exclusive table lock.
- If the **readpast** option is specified for an allpages-locked table, the **readpast** option is ignored. The command blocks as soon as it finds an incompatible lock.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored. The command executes at level 3. The command blocks on any rows or pages with incompatible locks.
- If the transaction isolation level for a session is 0, a **delete** command using **readpast** does not issue warning messages. For datapages-locked tables, **delete** with **readpast** modifies all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, it affects all rows that are not locked with incompatible locks.
- If the **delete** command applies to a row with two or more text columns, and any text column has an incompatible lock on it, **readpast** locking skips the row.

Using *index*, *prefetch*, or *lru | mru*

- The **index**, **prefetch**, and **lru | mru** options override the choices made by the Adaptive Server optimizer. Use these options with caution, and always check the performance impact with **set statistics io on**. For more information about using these options, see the *Performance and Tuning Guide*.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of more than one table in the **from** clause and qualification of table name with database name are Transact-SQL extensions. **readpast** is a Transact-SQL extension

Permissions

delete permission defaults to the table or view owner, who can transfer it to other users.

If **set ansi_permissions** is **on**, you must have **select** permission on all columns appearing in the **where** clause, in addition to the regular permissions required for **delete** statements. By default, **ansi_permissions** is **off**.

See Also

Commands	**create trigger**, **drop table**, **drop trigger**, **truncate table**, **where Clause**

delete statistics

Function

Removes statistics from the *sysstatistics* system table.

Syntax

```
delete [shared] statistics table_name [(column_name
   [, column_name]...)]
```

Parameters

shared – removes simulated statistics information from *sysstatistics* in the *master* database.

table_name – removes statistics for all columns in the table.

column_name – removes statistics for the specified column.

Examples

1. `delete statistics titles`

   Delete the densities, selectivities, and histograms for all columns in the *titles* table.

2. `delete statistics titles(pub_id)`

   Deletes densities, selectivities, and histograms for the *pub_id* column in the *titles* table.

3. `delete statistics titles(pub_id, pubdate)`

   Deletes densities, selectivities, and histograms for *pub_id*, *pubdate*, without affecting statistics on the single-column *pub_id* or the single-column *pubdate*.

Comments

- **delete statistics** removes statistics for the specified columns or table from the *sysstatistics* table. It does not affect statistics in the *systabstats* table.
- When you issue the **drop table** command, the corresponding rows in *sysstatistics* are dropped. When you use the **drop index** command, the rows in *sysstatistics* are not deleted. This allows the query optimizer to continue to use index statistics without incurring the overhead of maintaining the index on the table.

◆ ***WARNING!***

Densities, selectivities and histograms are essential to good query optimization. The delete statistics command is provided as a tool to remove statistics not used by the optimizer. If you inadvertently delete statistics needed for query optimization, run update statistics on the table, index or column.

- Loading simulated statistics with the **optdiag** utility command adds a small number of rows to *master..sysstatistics* table. If the simulated statistics are no longer in use, the information in *master..sysstatistics* can be dropped with the **delete shared statistics** command.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the table owner or a System Administrator can use **delete statistics**.

See Also

Commands	create index, update
Utility Program	optdiag

disk init

Function

Makes a physical device or file usable by Adaptive Server.

Syntax

```
disk init
   name = "device_name" ,
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number ]
   [, contiguous]
```

Keywords and Options

name – is the name of the database device or file. The name must conform to the rules for identifiers and must be enclosed in single or double quotes. This name is used in the **create database** and **alter database** commands.

physname – is the full specification of the database device. This name must be enclosed in single or double quotes.

vdevno – is the virtual device number. It must be unique among the database devices associated with Adaptive Server. The device number 0 is reserved for the master device. Valid device numbers are between 1 and 255, but the highest number must be one less than the number of database devices for which your Adaptive Server is configured. For example, for an Adaptive Server with the default configuration of 10 devices, the available device numbers are 1–9. To see the maximum number of devices available on Adaptive Server, run **sp_configure**, and check the **number of devices** value.

To determine the virtual device number, look at the *device_number* column of the **sp_helpdevice** report, and use the next unused integer.

size – is the size of the database device in 2K blocks.

If you plan to use the new device for the creation of a new database, the minimum **size** is the size of the *model* database, 1024 2K blocks (2MB). If you are initializing a log device, the **size** can

be as small as 512 2K blocks (1MB). The maximum size is system-dependent.

➤ ***Note***

The **disk init** command fails if the number of 2K blocks on the physical device is less than the sum of **size** and **vstart**.

vstart – is the starting virtual address, or the starting offset, in 2K blocks. The value for **vstart** should be 0 (the default) unless you are running the Logical Volume Manager on an AIX operating system, in which case, *vstart* should be 2.

Specify **vstart** only if instructed to do so by Sybase Technical Support.

cntrltype – specifies the disk controller. Its default value is 0. Reset **cntrltype** only if instructed to do so by Sybase Technical Support.

contiguous – (OpenVMS only) forces contiguous database file creation. This option is meaningful only when you are initializing a **file**; it has no effect when initializing a **foreign device**. If you include the **contiguous** option, the system creates a contiguous file or the command fails with an error message. If you do not include the **contiguous** option, the system still tries to create a contiguous file. If the system fails to create the file contiguously, it creates a file that does not force contiguity. In either case, the system displays a message indicating the type of file that is created.

Examples

```
1. disk init
     name = "user_disk",
     physname = "/dev/rxy1a",
     vdevno = 2, size = 5120
```

Initializes 5MB of a disk on a UNIX system.

```
2. disk init
    name = "user_disk",
    physname = "disk$rose_1:[dbs]user.dbs",
    vdevno = 2, size = 5120,
    contiguous
```

Initializes 5MB of a disk on an OpenVMS system, forcing the database file to be created contiguously.

Comments

- The master device is initialized by the installation program; it is not necessary to initialize this device with **disk init**.
- To successfully complete disk initialization, the "sybase" user must have the appropriate operating system permissions on the device that is being initialized.
- Use **disk init** for each new database device. Each time **disk init** is issued, a row is added to *master..sysdevices*. A new database device does not automatically become part of the pool of default database storage. Assign default status to a database device with the system procedure **sp_diskdefault**.
- On OpenVMS systems, using a logical name to refer to the **physname** offers more flexibility than using a hard-coded path name. For example, if you define the logical name ''userdisk'' as:

  ```
  disk$rose_1:[dbs]user.dbs
  ```

  you can change the **physname** in the example 2 above to "userdisk". To reorganize your disk or to move ''user.dbs'', just redefine the logical name as the new path.

  Any logical name used by an Adaptive Server must be:

  - A system logical name, or
  - A process logical name defined in the runserver file for that Adaptive Server.
- Back up the *master* database with the **dump database** or **dump transaction** command after each use of **disk init**. This makes recovery easier and safer in case *master* is damaged. (If you add a device with **disk init** and fail to back up *master*, you may be able to recover the changes by using **disk reinit**, then stopping and restarting Adaptive Server.)
- Assign user databases to database devices with the **on** ***device_name*** clause of the **create database** or **alter database** command.
- The preferred method for placing a database's transaction log (the system table *syslogs*) on a different device than the one on which the rest of the database is stored, is to use the **log on** extension to **create database**. Alternatively, you can name at least two devices when you create the database, then execute **sp_logdevice**. You can also use **alter database** to extend the database onto a second device, then run **sp_logdevice**. The **log on** extension immediately moves the entire log to a separate device. The

sp_logdevice method retains part of the system log on the original database device until transaction activity causes the migration to become complete.

- For a report on all Adaptive Server devices on your system (both database and dump devices), execute the system procedure **sp_helpdevice.**
- Remove a database device with the system procedure **sp_dropdevice**. You must first drop all existing databases on that device.

  After dropping a database device, you can create a new one with the same name (using **disk init**), as long as you give it a different physical name and virtual device number. If you want to use the same physical name and virtual device number, you must restart Adaptive Server.
- If **disk init** failed because the **size** value is too large for the database device, use a different virtual device number or restart Adaptive Server before executing **disk init** again.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

disk init permission defaults to System Administrators and is not transferable. You must be using the *master* database to use **disk init.**

See Also

Commands	**alter database**, **create database**, **disk refit**, **disk reinit**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_diskdefault**, **sp_dropdevice**, **sp_helpdevice**, **sp_logdevice**

disk mirror

Function

Creates a software mirror that immediately takes over when the primary device fails.

Syntax

```
disk mirror
   name = "device_name" ,
   mirror = "physicalname"
   [ ,writes = { serial | noserial }]
   [ ,contiguous ] (OpenVMS only)
```

Keywords and Options

name – is the name of the database device that you want to mirror. This is recorded in the *name* column of the *sysdevices* table. The name must be enclosed in single or double quotes.

mirror – is the full path name of the database mirror device that is to be your secondary device. It must be enclosed in single or double quotes. If the secondary device is a file, *physicalname* should be a path specification that clearly identifies the file, which Adaptive Server will create. It cannot be an existing file.

writes – allows you to choose whether to enforce serial writes to the devices. In the default case (**serial**), the write to the primary database device is guaranteed to finish before the write to the secondary device begins. If the primary and secondary devices are on different physical devices, serial writes can ensure that at least one of the disks will be unaffected in the event of a power failure.

contiguous – (OpenVMS only) is meaningful only if the mirror is a file rather than a foreign device. This option forces the file that will be used as the secondary device to be created contiguously. If you include the **contiguous** option, the system creates a contiguous file or the command fails with an error message. If you do not include the **contiguous** option, the system still tries to create a contiguous file. If it fails to create the file contiguously, the system creates a file that does not force contiguity. In either case, the system displays a message indicating the type of file that is created. The **contiguous** option is also available with **disk init** for OpenVMS users.

Examples

```
1. disk mirror
     name = "user_disk",
     mirror = "/server/data/mirror.dat"
```

Creates a software mirror for the database device *user_disk* on the file *mirror.dat*.

Comments

- Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

  Disk mirroring does not interfere with ongoing activities in the database. You can mirror or unmirror database devices without shutting down SQL Server.

- Back up the *master* database with the **dump database** command after each use of **disk mirror**. This makes recovery easier and safer in case *master* is damaged.

- When a read or write to a mirrored device is unsuccessful, Adaptive Server unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. The System Administrator must use the **disk remirror** command to restart mirroring.

- You can mirror the master device, devices that store data, and devices that store transaction logs. However, you cannot mirror dump devices.

- Devices are mirrored; databases are not.

- A device and its mirror constitute one logical device. Adaptive Server stores the physical name of the mirror device in the *mirrorname* column of the *sysdevices* table. It does not require a separate entry in *sysdevices* and should not be initialized with **disk init**.

- To retain use of asynchronous I/O, always mirror devices that are capable of asynchronous I/O to other devices capable of asynchronous I/O. In most cases, this means mirroring raw devices to raw devices and operating system files to operating system files.

  If the operating system cannot perform asynchronous I/O on files, mirroring a raw device to a regular file produces an error

message. Mirroring a regular file to a raw device will work, but will not use asynchronous I/O.

- Mirror all default database devices so that you are still protected if a **create** or **alter database** command affects a database device in the default list.
- For greater protection, mirror the database device used for transaction logs.
- Always put user database transaction logs on a separate database device. To put a database's transaction log (that is, the system table *syslogs*) on a device other than the one on which the rest of the database is stored, name the database device and the log device when you create the database. Alternatively, use **alter database** to extend the database onto a second device, then run **sp_logdevice**.
- If you mirror the database device for the *master* database, you can use the **-r** option and the name of the mirror for UNIX, or the **mastermirror** option for OpenVMS, when you restart Adaptive Server with the **dataserver** utility program. Add this to the *RUN_servername* file for that server so that the **startserver** utility program knows about it. For example:

  ```
  dataserver -dmaster.dat -rmirror.dat
  ```

  starts a master device named *master.dat* and its mirror, *mirror.dat*. For more information, see **dataserver** and **startserver** in the *Utility Programs* manual for your platform.
- If you mirror a database device that has unallocated space (room for additional **create database** and **alter database** statements to allocate part of the device), **disk mirror** begins mirroring these allocations when they are made, not when the **disk mirror** command is issued.
- For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute the system procedure **sp_helpdevice**.
- For more details about disk mirroring in the OpenVMS environment, see configuration documentation for your platform.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

disk mirror permission defaults to the System Administrator and is not transferable. You must be using the *master* database in order to use **disk mirror.**

See Also

Commands	**alter database**, **create database**, **disk init**, **disk refit**, **disk reinit**, **disk remirror**, **disk unmirror**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_diskdefault**, **sp_helpdevice**, **sp_logdevice**
Utility programs	**dataserver**, **startserver**

disk refit

Function

Rebuilds the *master* database's *sysusages* and *sysdatabases* system tables from information contained in *sysdevices*.

Syntax

```
disk refit
```

Examples

```
1. disk refit
```

Comments

- Adaptive Server automatically shuts down after **disk refit** rebuilds the system tables.
- Use **disk refit** after **disk reinit** as part of the procedure to restore the master database. For more information, see the *System Administration Guide*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

disk refit permission defaults to System Administrators and is not transferable. You must be in the *master* database to use **disk refit**.

See Also

Commands	**disk init**, **disk reinit**
System procedures	**sp_addumpdevice**, **sp_helpdevice**

disk reinit

Function

Rebuilds the *master* database's *sysdevices* system table. Use **disk reinit** as part of the procedure to restore the *master* database.

Syntax

```
disk reinit
   name = "device_name",
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number]
```

Keywords and Options

name – is the name of the database device. It must conform to the rules for identifiers, and it must be enclosed in single or double quotes. This name is used in the **create database** and **alter database** commands.

physname – is the name of the database device. The physical name must be enclosed in single or double quotes.

vdevno – is the virtual device number. It must be unique among devices used by Adaptive Server. The device number 0 is reserved for the *master* database device. Legal numbers are between 1 and 255, but cannot be greater than the number of database devices for which your system is configured. The default is 50 devices.

size – is the size of the database device in 2K blocks. The minimum usable size is 1024 2K blocks (2MB).

vstart – is the starting virtual address, or the starting offset, in 2K blocks. The value for **vstart** should be 0 (the default) unless you are running the Logical Volume Manager on an AIX operating system, in which case, *vstart* should be 2.

Specify **vstart** only if instructed to do so by Sybase Technical Support.

cntrltype – specifies the disk controller. Its default value is 0. Reset it only if instructed to do so by Sybase Technical Support.

Examples

```
1. disk reinit
     name = "user_disk",
     physname = "/server/data/userdata.dat",
     vdevno = 2, size = 5120
```

Comments

- **disk reinit** ensures that *master..sysdevices* is correct if the master database has been damaged or if devices have been added since the last dump of *master*.
- **disk reinit** is similar to **disk init**, but does not initialize the database device.
- For complete information on restoring the *master* database, see the *System Administration Guide*.

Standards and Compliance

Standard	Compliance level
SQL92	Transact-SQL extension

Permissions

disk reinit permission defaults to System Administrators and is not transferable. You must be in the *master* database to use **disk reinit**.

See Also

Commands	**alter database**, **create database**, **dbcc**, **disk init**, **disk refit**
System procedures	**sp_addumpdevice**, **sp_helpdevice**

disk remirror

Function

Restarts disk mirroring after it is stopped by failure of a mirrored device or temporarily disabled by the **disk unmirror** command.

Syntax

```
disk remirror
   name = "device_name"
```

Keywords and Options

name – is the name of the database device that you want to remirror. This is recorded in the *name* column of the *sysdevices* table. The name must be enclosed in single or double quotes.

Examples

```
1. disk remirror
     name = "user_disk"
```

Resumes software mirroring on the database device *user_disk*.

Comments

- Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

  Use the **disk remirror** command to reestablish mirroring after it has been temporarily stopped by failure of a mirrored device or temporarily disabled with the **mode = retain** option of the **disk unmirror** command. The **disk remirror** command copies data on the retained disk to the mirror.
- It is important to back up the *master* database with the **dump database** command after each use of **disk remirror.** This makes recovery easier and safer in case *master* is damaged.
- If mirroring was permanently disabled with the **mode = remove** option, you must remove the operating system file that contains the mirror before using **disk remirror.**
- Database devices, not databases, are mirrored.

- You can mirror, remirror, or unmirror database devices without shutting down Adaptive Server. Disk mirroring does not interfere with ongoing activities in the database.
- When a read or write to a mirrored device is unsuccessful, Adaptive Server unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. The System Administrator must use **disk remirror** to restart mirroring.
- In addition to mirroring user database devices, always put user database transaction logs on a separate database device. The database device used for transaction logs can also be mirrored for even greater protection. To put a database's transaction log (that is, the system table *syslogs*) on a different device than the one on which the rest of the database is stored, name the database device and the log device when you create the database. Alternatively, **alter database** to a second device, then run **sp_logdevice**.
- If you mirror the database device for the *master* database, you can use the **-r** option and the name of the mirror for UNIX, or the **mastermirror** option for OpenVMS, when you restart Adaptive Server with the **dataserver** utility program. Add this option to the *RUN_servername* file for that server so that the **startserver** utility program knows about it. For example:

  ```
  dataserver -dmaster.dat -rmirror.dat
  ```

  starts a master device named *master.dat* and its mirror, *mirror.dat*. For more information, see **dataserver** and **startserver** in the *Utility Programs* manual for your platform.
- For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute the system procedure **sp_helpdevice**.
- For more details about disk mirroring in the OpenVMS environment, see your configuration documentation for your platform.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

disk remirror permission defaults to the System Administrator and is not transferable. You must be using the *master* database to use **disk remirror.**

See Also

Commands	**alter database**, **create database**, **disk init**, **disk mirror**, **disk refit**, **disk reinit**, **disk unmirror**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_diskdefault**, **sp_helpdevice**, **sp_logdevice**
Utility programs	**dataserver**, **startserver**

disk unmirror

Function

Suspends disk mirroring initiated with the **disk mirror** command to allow hardware maintenance or the changing of a hardware device.

Syntax

```
disk unmirror
   name = "device_name"
   [ ,side = { "primary" | secondary }]
   [ ,mode = { retain | remove }]
```

Keywords and Options

name – is the name of the database device that you want to unmirror. The name must be enclosed in single or double quotes.

side – specifies whether to disable the **primary** device or the **secondary** device (the mirror). By default, the secondary device is unmirrored.

mode – determines whether the unmirroring is temporary (**retain**) or permanent (**remove**). By default, unmirroring is temporary.

Specify **retain** when you plan to remirror the database device later in the same configuration. This option mimics what happens when the primary device fails:

- I/O is directed only at the device **not** being unmirrored
- The *status* column of *sysdevices* indicates that mirroring is deactivated

remove eliminates all *sysdevices* references to a mirror device:

- The *status* column indicates that the mirroring feature is ignored
- The *phyname* column is replaced by the name of the secondary device in the *mirrorname* column if the primary device is the one being deactivated
- The *mirrorname* column is set to NULL

Examples

1. **disk unmirror**
   name = "user_disk"

   Suspends software mirroring for the database device *user_disk*.

2. **disk unmirror name = "user_disk", side = secondary**

   Suspends software mirroring for the database device *user_disk* on the secondary side.

3. **disk unmirror name = "user_disk", mode = remove**

   Suspends software mirroring for the database device *user_disk* and removes all device references to the mirror device.

Comments

- Disk mirroring creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

  disk unmirror disables either the original database device or the mirror, either permanently or temporarily, so that the device is no longer available to Adaptive Server for reads or writes. It does not remove the associated file from the operating system.

- Disk unmirroring alters the *sysdevices* table in the *master* database. It is important to back up the *master* database with the **dump database** command after each use of **disk unmirror**. This makes recovery easier and safer in case *master* is damaged.

- You can unmirror a database device while it is in use.

- You cannot unmirror any of a database's devices while a **dump database**, **load database**, or **load transaction** is in progress. Adaptive Server displays a message asking whether to abort the dump or load or to defer the **disk unmirror** until after the dump or load completes.

- You cannot unmirror a database's log device while a **dump transaction** is in progress. Adaptive Server displays a message asking whether to abort the dump or defer the **disk unmirror** until after the dump completes.

➤ ***Note***

dump transaction with truncate_only and **dump transaction with no_log** are not affected when a log device is unmirrored.

- You should mirror all the default database devices so that you are still protected if a **create** or **alter database** command affects a database device in the default list.
- When a read or write to a mirrored device is unsuccessful, Adaptive Server automatically unmirrors the bad device and prints error messages. Adaptive Server continues to run, unmirrored. A System Administrator must restart mirroring with the **disk remirror** command.
- For a report on all Adaptive Server devices on your system (user database devices and their mirrors, as well as dump devices), execute the system procedure **sp_helpdevice**.
- For more details about disk mirroring in the OpenVMS environment, see your configuration documentation for your platform.
- Use **disk remirror** to reestablish mirroring after it is temporarily stopped with the **mode = retain** option of the **disk unmirror** command. If mirroring is permanently disabled with the **mode = remove** option, you must remove the operating system file that contains the mirror before using **disk remirror**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

disk unmirror permission defaults to the System Administrator, and is not transferable. You must be using the *master* database to use **disk unmirror**.

See Also

Commands	**alter database**, **create database**, **disk init**, **disk mirror**, **disk refit**, **disk reinit**, **disk remirror**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_diskdefault**, **sp_helpdevice**, **sp_logdevice**
Utility programs	**dataserver**, **startserver**

drop database

Function

Removes one or more databases from Adaptive Server.

Syntax

```
drop database database_name [, database_name]...
```

Keywords and Options

database_name – is the name of a database to remove. Use **sp_helpdb** to get a list of databases.

Examples

```
1. drop database publishing
2. drop database publishing, newpubs
```

The dropped databases (and their contents) are gone.

Comments

- Removing a database deletes the database and all its objects, frees its storage allocation, and erases its entries from the *sysdatabases* and *sysusages* system tables in the *master* database.
- **drop database** clears the suspect page entries pertaining to the dropped database from *master..sysattributes*.

Restrictions

- You must be using the *master* database to drop a database.
- You cannot drop a database that is in use (open for reading or writing by any user).
- You cannot use **drop database** to remove a database that is referenced by a table in another database. Execute the following query to determine which tables and external databases have foreign key constraints on primary key tables in the current database:

  ```
  select object_name(tableid), db_name(frgndbname)
  from sysreferences
  where frgndbname is not null
  ```

  Use **alter table** to drop these cross-database constraints, then reissue the **drop database** command.

- You cannot use **drop database** to remove a damaged database. Use the **dbcc dbrepair** command:

  dbcc dbrepair (*database_name*, dropdb)

- You cannot drop the *sybsecurity* database if auditing is enabled. When auditing is disabled, only the System Security Officer can drop *sybsecurity*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the Database Owner can execute **drop database**, except for the *sybsecurity* database, which can be dropped only by the System Security Officer.

See Also

Commands	**alter database**, **create database**, **dbcc**, **use**
System procedures	**sp_changedbowner**, **sp_helpdb**, **sp_renamedb**, **sp_spaceused**

drop default

Function

Removes a user-defined default.

Syntax

```
drop default [owner.]default_name
   [, [owner.]default_name]...
```

Keywords and Options

default_name – is the name of an existing default. Execute **sp_help** to get a list of existing defaults. Specify the owner's name to drop a default of the same name owned by a different user in the current database. The default value for *owner* is the current user.

Examples

```
1. drop default datedefault
```

Removes the user-defined default *datedefault* from the database.

Comments

- You cannot drop a default that is currently bound to a column or to a user-defined datatype. Use the system procedure **sp_unbindefault** to unbind the default before you drop it.
- You can bind a new default to a column or user-defined datatype without unbinding its current default. The new default overrides the old one.
- When you drop a default for a NULL column, NULL becomes the column's default value. When you drop a default for a NOT NULL column, an error message appears if users do not explicitly enter a value for that column when inserting data.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop default permission defaults to the owner of the default and is not transferable.

See Also

Commands	**create default**
System procedures	**sp_help**, **sp_helptext**, **sp_unbindefault**

drop index

Function

Removes an index from a table in the current database.

Syntax

```
drop index table_name.index_name
   [, table_name.index_name]...
```

Keywords and Options

table_name – is the table in which the indexed column is located. The table must be in the current database.

index_name – is the index to drop. In Transact-SQL, index names need not be unique in a database, though they must be unique within a table.

Examples

1.
```
drop index authors.au_id_ind
```

The index *au_id_ind* in the *authors* table no longer exists.

Comments

- Once the **drop index** command is issued, you regain all the space that was previously occupied by the index. This space can be used for any database objects.
- You cannot use **drop index** on system tables.
- **drop index** cannot remove indexes that support unique constraints. To drop such indexes, drop the constraints through **alter table** or drop the table. See **create table** for more information about unique constraint indexes.
- You cannot drop indexes that are currently used by any open cursor. For information about which cursors are open and what indexes they use, use **sp_cursorinfo**.
- To get information about what indexes exist on a table, use:

  ```
  sp_helpindex objname
  ```

  where *objname* is the name of the table.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop index permission defaults to the index owner and is not transferable.

See Also

Commands	**create index**
System procedures	**sp_cursorinfo**, **sp_helpindex**, **sp_spaceused**

drop procedure

Function

Removes a procedure.

Syntax

```
drop proc[edure] [owner.]procedure_name
   [, [owner.]procedure_name] ...
```

Keywords and Options

procedure_name – is the name of the procedure to drop. Specify the owner's name to drop a procedure of the same name owned by a different user in the current database. The default value for *owner* is the current user.

Examples

1. **drop procedure showind**

   Deletes the stored procedure **showind**.

2. **drop procedure xp_echo**

   Unregisters the extended stored procedure *xp_echo*.

Comments

- **drop procedure** drops user-defined stored procedures, system procedures, and extended stored procedures (ESPs).
- Adaptive Server checks the existence of a procedure each time a user or a program executes that procedure.
- A procedure group (more than one procedure with the same name but with different;*number* suffixes) can be dropped with a single **drop procedure** statement. For example, if the procedures used with the application named **orders** were named *orderproc;1*, *orderproc;2*, and so on, the following statement:

  drop proc orderproc

  drops the entire group. Once procedures have been grouped, individual procedures within the group cannot be dropped. For example, the statement:

  drop procedure orderproc;2

  is not allowed.

You cannot drop extended stored procedures as a procedure group.

- The system procedure **sp_helptext** displays the procedure's text, which is stored in *syscomments*.
- The system procedure **sp_helpextendedproc** displays ESPs and their corresponding DLLs.
- Dropping an ESP unregisters the procedure by removing it from the system tables. It has no effect on the underlying DLL.
- **drop procedure** drops only user-created procedures from your current database.

Standards and Compliance

Standard	Compliance level
SQL92	Transact-SQL extension

Permissions

drop procedure permission defaults to the procedure owner and is not transferable.

See Also

Commands	create procedure
System procedures	sp_depends, sp_dropextendedproc, sp_helpextendedproc, sp_helptext, sp_rename

drop role

Function

Drops a user-defined role.

Syntax

```
drop role role_name [with override]
```

Keywords and Options

role_name – is the name of the role you want to drop.

with override – overrides any restrictions on dropping a role. When you use the **with override** option, you can drop any role without having to check whether the role permissions have been dropped in each database.

Examples

1. `drop role doctor_role`

   Drops the named role only if all permissions in all databases have been revoked. The System Administrator or object owner must revoke permissions granted in each database before dropping a role, or the command fails.

2. `drop role doctor_role with override`

   Drops the named role and removes permission information and any other reference to the role from all databases.

Comments

- You need not drop memberships before dropping a role. Dropping a role automatically removes any user's membership in that role, regardless of whether you use the **with override** option.
- Use **drop role** from the *master* database.

Restrictions

- You cannot use **drop role** to drop system roles.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Security Officer to use **drop role**.

drop role permission is not included in the **grant all** command.

See Also

Commands	**alter role**, **create role**, **grant**, **revoke**, **set**
System procedures	**sp_activeroles**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**

drop rule

Function

Removes a user-defined rule.

Syntax

```
drop rule [owner.]rule_name [, [owner.]rule_name]...
```

Examples

```
1. drop rule pubid_rule
```

Removes the rule *pubid_rule* from the current database.

Keywords and Options

rule_name – is the name of the rule to drop. Specify the owner's name to drop a rule of the same name owned by a different user in the current database. The default value for *owner* is the current user.

Comments

- Before dropping a rule, you must unbind it using the system procedure **sp_unbindrule.** If the rule has not been unbound, an error message appears, and the **drop rule** command fails.
- You can bind a new rule to a column or user-defined datatype without unbinding its current rule. The new rule overrides the old one.
- After you drop a rule, Adaptive Server enters new data into the columns that were previously governed by the rule without constraints. Existing data is not affected in any way.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop rule permission defaults to the rule owner and is not transferable.

See Also

Commands	**create rule**
System procedures	**sp_bindrule**, **sp_help**, **sp_helptext**, **sp_unbindrule**

drop table

Function

Removes a table definition and all of its data, indexes, triggers, and permissions from the database.

Syntax

```
drop table [[database.]owner.]table_name
   [, [[database.]owner.]table_name ]...
```

Keywords and Options

table_name – is the name of the table to drop. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

Examples

```
1. drop table roysched
```

Removes the table *roysched* and its data and indexes from the current database.

Comments

- When you use **drop table**, any rules or defaults on the table lose their binding, and any triggers associated with it are automatically dropped. If you re-create a table, you must rebind the appropriate rules and defaults and re-create any triggers.
- The system tables affected when a table is dropped are *sysobjects*, *syscolumns*, *sysindexes*, *sysprotects*, and *syscomments*.
- If Component Integration Services is enabled, and if the table being dropped was created with **create existing table**, the table is not dropped from the remote server. Instead, Adaptive Server removes references to the table from the system tables.

Restrictions

- You cannot use the **drop table** command on system tables.
- Once you have partitioned a table, you cannot drop it. You must use the **unpartition** clause of the **alter table** command before you can issue the **drop table** command.

- You can drop a table in any database, as long as you are the table owner. For example, to drop a table called *newtable* in the database *otherdb*:

  ```
  drop table otherdb..newtable
  ```

  or:

  ```
  drop table otherdb.yourname.newtable
  ```

- If you **delete** all the rows in a table or use the **truncate table** command, the table still exists until you **drop** it.

Dropping Tables with Cross-Database Referential Integrity Constraints

- When you create a cross-database constraint, Adaptive Server stores the following information in the *sysreferences* system table of each database:

Table 6-21: Information stored about referential integrity constraints

Information Stored in *sysreferences*	Columns with Information About Referenced Table	Columns with Information About Referencing Table
Key Column IDs	*refkey1* through *refkey16*	*fokey1* through *fokey16*
Table ID	*reftabid*	*tableid*
Database Name	*pmrydbname*	*frgndbname*

- Because the referencing table depends on information from the referenced table, Adaptive Server does not allow you to:
  - Drop the referenced table,
  - Drop the external database that contains it, or
  - Rename either database with **sp_renamedb.**

  Use the **sp_helpconstraint** system procedure to determine which tables reference the table you want to drop. Use **alter table** to drop the constraints before reissuing the **drop table** command.
- You can drop a referencing table or its database without problems. Adaptive Server automatically removes the foreign key information from the referenced database.
- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

- ***WARNING!***

  Loading earlier dumps of these databases could cause database corruption. For more information about loading databases with cross-database referential integrity constraints, see the *System Administration Guide*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop table permission defaults to the table owner and is not transferable.

See Also

Commands	**alter table**, **create table**, **delete**, **truncate table**
System procedures	**sp_depends**, **sp_help**, **sp_spaceused**

drop trigger

Function

Removes a trigger.

Syntax

```
drop trigger [owner.]trigger_name
   [, [owner.]trigger_name]...
```

Keywords and Options

trigger_name – is the name of the trigger to drop. Specify the owner's name to drop a trigger of the same name owned by a different user in the current database. The default value for *owner* is the current user.

Examples

```
1. drop trigger trigger1
```

Removes the trigger *trigger1* from the current database.

Comments

- **drop trigger** drops a trigger in the current database.
- You do not need to explicitly drop a trigger from a table in order to create a new trigger for the same operation (**insert**, **update**, or **delete**). In a table or column each new trigger for the same operation overwrites the previous one.
- When a table is dropped, Adaptive Server automatically drops any triggers associated with it.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop trigger permission defaults to the trigger owner and is not transferable.

See Also

Commands	**create trigger**
System procedures	**sp_depends**, **sp_help**, **sp_helptext**

drop view

Function

Removes one or more views from the current database.

Syntax

```
drop view [owner.]view_name [, [owner.]view_name]...
```

Keywords and Options

view_name – is the name of the view to drop. Specify the owner's name to drop a view of the same name owned by a different user in the current database. The default value for *owner* is the current user.

Examples

```
1. drop view new_price
```

Removes the view *new_price* from the current database.

Comments

- When you use **drop view**, the definition of the view and other information about it, including privileges, is deleted from the system tables *sysobjects*, *syscolumns*, *syscomments*, *sysdepends*, *sysprocedures*, and *sysprotects*.
- Existence of a view is checked each time the view is referenced, for example, by another view or by a stored procedure.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

drop view permission defaults to the view owner and is not transferable.

See Also

Commands	create view
System procedures	sp_depends, sp_help, sp_helptext

dump database

Function

Makes a backup copy of the entire database, including the transaction log, in a form that can be read in with **load database**. Dumps and loads are performed through Backup Server.

Syntax

```
dump database database_name
   to stripe_device [at backup_server_name]
      [density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name]
   [stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
        density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name,
        [dismount | nodismount],
        [nounload | unload],
        retaindays = number_days,
        [noinit | init],
        notify = {client | operator_console}
        } ]
```

Keywords and Options

database_name – is the name of the database from which you are copying data. The database name can be specified as a literal, a local variable, or a stored procedure parameter.

to *stripe_device* – is the device to which to copy the data. See "Specifying Dump Devices" in this section for information about what form to use when specifying a dump device.

at *backup_server_name* – is the name of the Backup Server. Do not specify this parameter when dumping to the default Backup Server. Specify this parameter only when dumping over the network to a remote Backup Server. You can specify up to 32 remote Backup Servers with this option. When dumping across the network, specify the *network name* of a remote Backup Server running on the machine to which the dump device is attached. For platforms that use interfaces files, the *backup_server_name* must appear in the interfaces file.

density = *density_value* – overrides the default density for a tape device. **Use this option only when reinitializing a volume on OpenVMS systems**. Valid densities are **800**, **1600**, **6250**, **6666**, **10000**, and **38000**. Not all values are valid for every tape drive; use the correct density for your tape drive.

blocksize = *number_bytes* – overrides the default block size for a dump device. The block size must be at least one database page (2048 bytes for most systems) and must be an exact multiple of the database page size. On OpenVMS systems, block size cannot exceed 55,296 bytes. Increasing the block size may improve the dump performance on some dump devices. For optimal performance, specify the blocksize as a power of 2, for example, 65536, 131072, or 262166.

capacity = *number_kilobytes* – is the maximum amount of data that the device can write to a single tape volume. The capacity must be at least five database pages and should be less than the recommended capacity for your device.

A general rule for calculating capacity is to use 70 percent of the manufacturer's maximum capacity for the device, allowing 30 percent for overhead such as inter-record gaps and tape marks. The maximum capacity is the capacity of the device on the drive, not the drive itself. This rule works in most cases, but may not work in all cases due to differences in overhead across vendors and across devices.

On UNIX platforms that cannot reliably detect the end-of-tape marker, indicate how many kilobytes can be dumped to the tape. You **must** supply a **capacity** for dump devices specified as a physical path name. If a dump device is specified as a logical

device name, the Backup Server uses the *size* parameter stored in the *sysdevices* system table unless you specify a capacity.

dumpvolume = *volume_name* – establishes the name that is assigned to the volume. The maximum length of *volume_name* is 6 characters. Backup Server writes the *volume_name* in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. The **load database** command checks the label and generates an error message if the wrong volume is loaded.

◆ ***WARNING!***

Be sure to label each tape volume as you create it so that the operator can load the correct tape.

stripe on *stripe_device* – is an additional dump device. You can use up to 32 devices, including the device named in the **to** *stripe_device* clause. The Backup Server splits the database into approximately equal portions, and sends each portion to a different device. Dumps are made concurrently on all devices, reducing the time required to make a dump and requiring fewer volume changes during the dump. See "Specifying Dump Devices" for information about how to specify a dump device.

dismount | **nodismount** – **on platforms, such as OpenVMS, that support logical dismount**, determines whether tapes remain mounted. By default, all tapes used for a dump are dismounted when the dump completes. Use **nodismount** to keep tapes available for additional dumps or loads.

nounload | **unload** – determines whether tapes rewind after the dump completes. By default, tapes do not rewind, allowing you to make additional dumps to the same tape volume. Specify **unload** for the last dump file to be added to a multidump volume. This rewinds and unloads the tape when the dump completes.

retaindays = *number_days* – when dumping to disk **on UNIX systems**, specifies the number of days that Backup Server protects you from overwriting the dump. If you try to overwrite the dump before it expires, Backup Server requests confirmation before overwriting the unexpired volume. **This option is meaningful only when dumping to a disk. It is not meaningful for tape dumps**.

The *number_days* must be a positive integer or 0, for dumps that you can overwrite immediately. If you do not specify a **retaindays** value, Backup Server uses the **tape retention in days** value set by **sp_configure**.

noinit | **init** – determines whether to append the dump to existing dump files or reinitialize (overwrite) the tape volume. By default, Adaptive Server appends dumps following the last end-of-tape mark, allowing you to dump additional databases to the same volume. New dumps can be appended only to the last volume of a multivolume dump. Use **init** for the first database you dump to a tape to overwrite its contents.

Use **init** when you want Backup Server to store or update tape device characteristics in the tape configuration file. For more information, see the *System Administration Guide*.

file = *file_name* – is the name of the dump file. The name cannot exceed 17 characters and must conform to operating system conventions for file names. For more information, see "Dump Files."

notify = {**client** | **operator_console**} – overrides the default message destination.

On operating systems that offer an operator terminal feature, such as OpenVMS, volume change messages are always sent to the operator terminal on the machine on which Backup Server is running. Use **client** to route other Backup Server messages to the terminal session that initiated the **dump database**.

On operating systems that do not offer an operator terminal feature, such as UNIX, messages are sent to the client that initiated the **dump database**. Use **operator_console** to route messages to the terminal on which Backup Server is running.

Examples

1. For OpenVMS:

```
dump database pubs2
  to "MTA0:"
```

For UNIX:

```
dump database pubs2
  to "/dev/nrmt0"
```

Dumps the database *pubs2* to a tape device. If the tape has an ANSI tape label, this command appends this dump to the files already on the tape, since the **init** option is not specified.

2. For OpenVMS:

```
dump database pubs2
        to "MTA0:" at REMOTE_BKP_SERVER
    stripe on "MTA1:" at REMOTE_BKP_SERVER
    stripe on "MTA2:" at REMOTE_BKP_SERVER
```

For UNIX:

```
dump database pubs2
    to "/dev/rmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
with retaindays = 14
```

Dumps the *pubs2* database, using the REMOTE_BKP_SERVER Backup Server. The command names three dump devices, so the Backup Server dumps approximately one-third of the database to each device. This command appends the dump to existing files on the tapes. On UNIX systems, the **retaindays** option specifies that the tapes cannot be overwritten for 14 days. (OpenVMS systems do not use the **retaindays** option; they always create new versions of files.)

3. For OpenVMS:

```
dump database pubs2
    to "MTA0:"
    with init
```

For UNIX:

```
dump database pubs2
    to "/dev/nrmt0"
    with init
```

The **init** option initializes the tape volume, overwriting any existing files.

4. For OpenVMS:

```
dump database pubs2
  to "MTA0:"
    with unload
```

For UNIX:

```
dump database pubs2
    to "/dev/nrmt0"
    with unload
```

Rewinds the dump volumes upon completion of the dump.

5. For OpenVMS:

```
dump database pubs2
  to "MTA0:"
    with notify = client
```

For UNIX:

```
dump database pubs2
    to "/dev/nrmt0"
      with notify = client
```

The **notify** clause sends Backup Server messages requesting volume changes to the client which initiated the dump request, rather than sending them to the default location, the console of the Backup Server machine.

Comments

- Table 6-22 describes the commands and system procedures used to back up databases:

Table 6-22: Commands used to back up databases and logs

To Do This	Use This Command
Make routine dumps of the entire database, including the transaction log.	**dump database**
Make routine dumps of the transaction log, then truncate the inactive portion.	**dump transaction**
Dump the transaction log after failure of a database device.	**dump transaction with no_truncate**
Truncate the log without making a backup, then copy the entire database.	**dump transaction with truncate_only** **dump database**
Truncate the log after your usual method fails due to insufficient log space, then copy the entire database.	**dump transaction with no_log** **dump database**
Respond to the Backup Server's volume change messages.	**sp_volchanged**

dump database Restrictions

- You cannot dump from an 11.x Adaptive Server to a 10.x Backup Server.
- You cannot have Sybase dumps and non-Sybase data (for example, UNIX archives) on the same tape.

- If a database has cross-database referential integrity constraints, the *sysreferences* system table stores the **name**—not the ID number—of the external database. Adaptive Server cannot guarantee referential integrity if you use **load database** to change the database name or to load it onto a different server.

◆ ***WARNING!***

Before dumping a database in order to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

- You cannot use **dump database** in a user-defined transaction.
- If you issue **dump database** on a database where a **dump transaction** is already in progress, **dump database** sleeps until the transaction dump completes.
- When using 1/4-inch cartridge tape, you can dump only one database or transaction log per tape.
- You cannot dump a database if it has offline pages. To force offline pages online, use **sp_forceonline_db** or **sp_forceonline_page.**

Scheduling Dumps

- Adaptive Server database dumps are **dynamic**—they can take place while the database is active. However, they may slow the system down slightly, so you may want to run **dump database** when the database is not being heavily updated.
- **Back up the *master* database regularly and frequently**. In addition to your regular backups, dump *master* after each **create database, alter database**, and **disk init** command is issued.
- Back up the *model* database each time you make a change to the database.
- Use **dump database** immediately after creating a database, to make a copy of the entire database. You cannot run **dump transaction** on a new database until you have run **dump database**.
- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of these databases could cause database corruption.

- Develop a regular schedule for backing up user databases and their transaction logs.
- Use thresholds to automate backup procedures. To take advantage of Adaptive Server's last-chance threshold, create user databases with log segments on a device that is separate from data segments. For more information about thresholds, see the *System Administration Guide*.

Dumping the System Databases

- The *master*, *model*, and *sybsystemprocs* databases do not have separate segments for their transaction logs. Use **dump transaction with truncate_only** to purge the log, then use **dump database** to back up the database.
- Backups of the *master* database are needed for recovery procedures in case of a failure that affects the *master* database. See the *System Administration Guide* for step-by-step instructions for backing up and restoring the *master* database.
- If you are using removable media for backups, the entire *master* database must fit on a single volume unless you have another Adaptive Server that can respond to volume change messages.

Specifying Dump Devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You cannot dump to the null device (on UNIX, */dev/null*; on OpenVMS, any device name beginning with "NL").
- Dumping to multiple stripes is supported for tape and disk devices. Placing multiple dumps on a device is supported only for tape devices.
- You can specify a local dump device as:
  - A logical device name from the *sysdevices* system table
  - An absolute path name
  - A relative path name

Backup Server resolves relative path names using Adaptive Server's current working directory.

- When dumping across the network, you must specify the absolute path name of the dump device. The path name must be valid on the machine on which Backup Server is running. If the name includes any characters except letters, numbers, or the underscore (_), you must enclose it in quotes.
- Ownership and permissions problems on the dump device may interfere with the use of **dump** commands. The **sp_addumpdevice** procedure adds the device to the system tables, but does not guarantee that you can dump to that device or create a file as a dump device.
- You can run more than one dump (or load) at the same time, as long as each uses different dump devices.
- If the device file already exists, Backup Server overwrites it; it does not truncate it. For example, suppose you dump a database to a device file and the device file becomes 10MB. If the next dump of the database to that device is smaller, the device file is still 10MB.

Determining Tape Device Characteristics

- If you issue a **dump** command without the **init** qualifier and Backup Server cannot determine the device type, the **dump** command fails. For more information, see the *System Administration Guide*.

Backup Servers

- You must have a Backup Server running on the same machine as Adaptive Server. (On OpenVMS systems, the Backup Server can be running in the same cluster as the Adaptive Server, as long as all database devices are visible to both.) The Backup Server must be listed in the *master..sysservers* table. This entry is created during installation or upgrade, and should not be deleted.
- If your backup devices are located on another machine so that you dump across a network, you must also have a Backup Server installed on the remote machine.

Dump Files

- Dumping a database with the **init** option overwrites any existing files on the tape or disk.

- Backup Server sends the dump file name to the location specified by the **with notify** clause. Before storing a backup tape, the operator should label it with the database name, file name, date, and other pertinent information. When loading a tape without an identifying label, use the **with headeronly** and **with listonly** options to determine the contents.

File Names and Archive Names

- The name of a dump file identifies the database that was dumped and when the dump was made. However, in the syntax:

  file = ***file_name***

  file_name has different meanings depending on whether you are dumping to disk or to a UNIX tape.

  In a dump to disk, the path name of a disk file is also its file name.

  In a dump to a UNIX tape, the path name is not the file name. The ANSI Standard Format for File Interchange contains a file name field in the HDR1 label. For tapes conforming to the ANSI specification, this field in the label identifies the file name. The ANSI specification only applies these labels to tape; it does not apply to disk files.

  This creates two problems:

  - UNIX does not follow the ANSI convention for tape file names. UNIX considers the tape's data to be unlabeled. Although it can be divided into files, those files have no name.
  - In Backup Server the ANSI tape labels are used to store information about the archive, negating the ANSI meanings. Therefore, disk files also have ANSI labels, because the archive name is stored there.

  The meaning of filename changes depending on the kind of dump you are performing. For example, in the following syntax:

```
dump database database_name to 'filename' with file='filename'
```

  - The first *filename* refers to the path name you enter to display the file.
  - The second *filename* is actually the archive name, the name stored in the HDR1 label in the archive, which the user can specify with the **file=***filename* parameter of the **dump** or **load** command.

When the archive name is specified, the server uses that name during a database load to locate the selected archive.

If the archive name is not specified, the server loads the first archive it encounters.

In both cases, file ='archivename' establishes the name that is stored in the HDR1 label, and which the subsequent LOAD will use to validate that it's looking at the right data.

If it is not specified, a DUMP will make one up and a LOAD will take the first one it finds.

The meaning of *filename* in the **to '*filename*'** clause changes according to whether this is a disk or tape dump:

- If the dump is to tape, 'filename' is the name of the tape device;
- If the dump is to disk, it is the name of a disk file.

If this is a disk dump and the 'filename' is not a complete path, it is modified by prepending the server's current working directory.

- If you are dumping to tape and you do not specify a file name, Backup Server creates a default file name by concatenating the following:
  - Last seven characters of the database name
  - Two-digit year number
  - Three-digit day of the year (1–366)
  - Hexadecimal-encoded time at which the dump file was created

  For example, the file *cations980590E100* contains a copy of the *publications* database made on the fifty-ninth day of 1998:

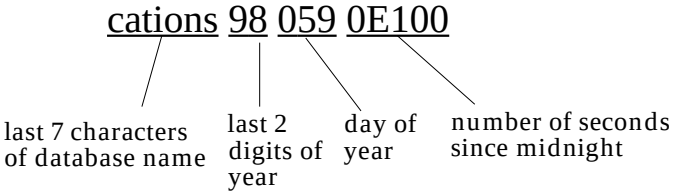

Figure 6-2: File naming convention for database dumps to tape

Volume Names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.
- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

➤ ***Note***

When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing Dump Volumes

- On OpenVMS systems, the operating system requests a volume change when it detects the end of a volume or when the specified drive is offline. After mounting another volume, the operator uses the **REPLY** command to reply to these messages.
- On UNIX systems, Backup Server requests a volume change when the tape capacity has been reached. After mounting another volume, the operator notifies Backup Server by executing the **sp_volchanged** system procedure on any Adaptive Server that can communicate with Backup Server.
- If Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. The operator responds to these messages with the **sp_volchanged** system procedure.

Appending to or Overwriting a Volume

- By default (**noinit**), Backup Server writes successive dumps to the same tape volume, making efficient use of high-capacity tape media. Data is added following the last end-of-tape mark. New dumps can be appended only to the last volume of a multivolume dump. Before writing to the tape, Backup Server verifies that the first file has not yet expired. If the tape contains non-Sybase data, Backup Server rejects it to avoid destroying potentially valuable information.
- Use the **init** option to reinitialize a volume. If you specify **init**, Backup Server overwrites any existing contents, even if the tape

contains non-Sybase data, the first file has not yet expired, or the tape has ANSI access restrictions.

- Figure 6-3 illustrates how to dump three databases to a single volume using:
  - **init** to initialize the tape for the first dump
  - **noinit** (the default) to append subsequent dumps
  - **unload** to rewind and unload the tape after the last dump

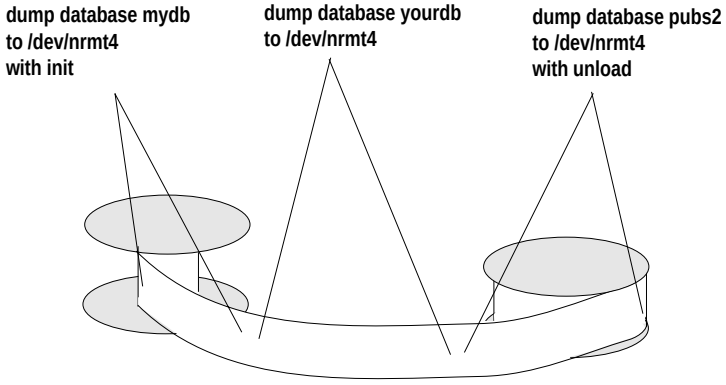

Figure 6-3: Dumping several databases to the same volume

Dumping Databases Whose Devices Are Mirrored

- At the beginning of a **dump database**, Adaptive Server passes Backup Server the primary device name of all database and log devices. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before the Backup Server completes its data transfer, Adaptive Server aborts the dump.
- If a user attempts to unmirror any of the named database devices while a **dump database** is in progress, Adaptive Server displays a message. The user executing the **disk unmirror** command can abort the dump or defer the **disk unmirror** until after the dump is complete.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the System Administrator, the Database Owner, and users with the Operator role can execute **dump database**.

See Also

Commands	dump transaction, load database, load transaction
System procedures	sp_addthreshold, sp_addumpdevice, sp_dropdevice, sp_dropthreshold, sp_helpdevice, sp_helpdb, sp_helpthreshold, sp_logdevice, sp_spaceused, sp_volchanged

dump transaction

Function

Makes a copy of a transaction log and removes the inactive portion.

Syntax

To make a routine log dump:

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       notify = {client | operator_console},
       standby_access }]
```

To truncate the log without making a backup copy:

```
dump tran[saction] database_name
   with truncate_only
```

To truncate a log that is filled to capacity. **Use only as a last resort**:

```
dump tran[saction] database_name
   with no_log
```

To back up the log after a database device fails:

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       no_truncate,
       notify = {client | operator_console}}]
```

Keywords and Options

database_name – is the name of the database from which you are copying data. The name can be given as a literal, a local variable, or a parameter to a stored procedure.

truncate_only – removes the inactive part of the log **without making a backup copy**. Use on databases without log segments on a separate device from data segments, Do not specify a dump device or Backup Server name.

no_log – removes the inactive part of the log **without making a backup copy and without recording the procedure in the transaction log**. Use **no_log** only when you have totally run out of log space and cannot run your usual **dump transaction** command. Use **no_log** as a last resort and use it only once after **dump transaction with truncate_only** fails. For additional information, see the *System Administration Guide.*

to *stripe_device* – is the device to which data is being dumped. See "Specifying Dump Devices" for information about what form to use when specifying a dump device.

at *backup_server_name* – is the name of the Backup Server. Do not specify this parameter if dumping to the default Backup Server. Specify this parameter only if dumping over the network to a remote Backup Server. You can specify up to 32 different remote Backup Servers using this option. When dumping across the network, specify the *network name* of a remote Backup Server running on the machine to which the dump device is attached. For platforms that use interfaces files, the *backup_server_name* must appear in the interfaces file.

density = *density_value* – overrides the default density for a tape device. **Use this option only when reinitializing a volume on OpenVMS systems**. Valid densities are **800**, **1600**, **6250**, **6666**, **10000**, and **38000**. Not all values are valid for every tape drive; use the correct density for your tape drive.

blocksize = *number_bytes* – overrides the default block size for a dump device. (**Wherever possible, use the default block size**; it is the best block size for your system.) The block size must be at least one database page (2048 bytes for most systems) and must be an exact multiple of the database page size. On OpenVMS systems, block size cannot exceed 55,296 bytes.

capacity = *number_kilobytes* – is the maximum amount of data that the device can write to a single tape volume. The capacity must be at least five database pages, and should be slightly less than the recommended capacity for your device.

A general rule for calculating capacity is to use 70 percent of the manufacturer's maximum capacity for the device, leaving 30 percent for overhead such as inter-record gaps and tape marks. This rule works in most cases, but may not work in all cases because of differences in overhead across vendors and devices.

OpenVMS systems write until they reach the physical end-of-tape marker, when they send a volume change request.

On UNIX platforms that cannot reliably detect the end-of-tape marker, you must indicate how many kilobytes can be dumped to the tape. You **must** supply a **capacity** for dump devices specified as a physical path name. If a dump device is specified as a logical device name, the Backup Server uses the *size* parameter stored in the *sysdevices* system table, unless you specify a capacity.

dumpvolume = *volume_name* – establishes the name that is assigned to the volume. The maximum length of *volume_name* is 6 characters. The Backup Server writes the *volume_name* in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. The **load transaction** command checks the label and generates an error message if the wrong volume is loaded.

stripe on *stripe_device* – is an additional dump device. You can use up to 32 devices, including the device named in the **to** *stripe_device* clause. The Backup Server splits the log into approximately equal portions and sends each portion to a different device. Dumps are made concurrently on all devices, reducing the time and the number of volume changes required. See "Specifying Dump Devices" for information about how to specify a dump device.

dismount | **nodismount** – **on platforms that support logical dismount** (such as OpenVMS), determines whether tapes remain mounted. By default, all tapes used for a dump are dismounted when the dump completes. Use **nodismount** to keep tapes available for additional dumps or loads.

nounload | **unload** – determines whether tapes rewind after the dump completes. By default, tapes do not rewind, allowing you to make additional dumps to the same tape volume. Specify **unload** for the last dump file to be added to a multidump volume. This rewinds and unloads the tape when the dump completes.

retaindays = *number_days* – **on UNIX systems**, specifies the number of days that Backup Server protects you from overwriting a dump. **This option is meaningful for disk, 1/4-inch cartridge, and single-file media. On multifile media, this option is meaningful for all volumes but the first**. If you try to overwrite a dump before it expires, Backup Server requests confirmation before overwriting the unexpired volume.

The *number_days* must be a positive integer or 0, for dumps you can overwrite immediately. If you do not specify a **retaindays** value, Backup Server uses the server-wide **tape retention in days** value, set by **sp_configure.**

noinit | **init** – determines whether to append the dump to existing dump files or reinitialize (overwrite) the tape volume. By default, Adaptive Server appends dumps following the last end-of-tape mark, allowing you to dump additional databases to the same volume. New dumps can be appended only to the last volume of a multivolume dump. Use **init** for the first database you dump to a tape, to overwrite its contents.

Use **init** when you want Backup Server to store or update tape device characteristics in the tape configuration file. For more information, see the *System Administration Guide.*

file = *file_name* – is the name of the dump file. The name cannot exceed 17 characters and must conform to operating system conventions for file names. If you do not specify a file name, Backup Server creates a default file name. For more information, see "Dump Files".

no_truncate – dumps a transaction log, **even if the disk containing the data segments for a database is inaccessible**, using a pointer to the transaction log in the *master* database. The **with no_truncate** option provides up-to-the-minute log recovery when the transaction log resides on an undamaged device, and the *master* database and user databases reside on different physical devices.

notify = {**client** | **operator_console**} – overrides the default message destination.

- On operating systems (such as OpenVMS) that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use **client** to route other Backup Server messages to the terminal session that initiated the **dump database**.
- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that initiated the **dump database**. Use **operator_console** to route messages to the terminal on which the Backup Server is running.

with standby_access – specifies that only completed transactions are to be dumped. The dump continues to the furthest point it can find

at which a transaction has just completed and there are no other active transactions.

Examples

1. **For UNIX:**

   ```
   dump transaction pubs2
     to "/dev/nrmt0"
   ```

   For OpenVMS:

   ```
   dump database pubs2
     to "MTA0:"
   ```

   Dumps the transaction log to a tape, appending it to the files on the tape, since the **init** option is not specified.

2. **For UNIX:**

   ```
   dump transaction mydb
       to "/dev/nrmt4" at REMOTE_BKP_SERVER
       stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
   with init, retaindays = 14
   ```

   For OpenVMS:

   ```
   dump transaction mydb
       to "MTA0:" at REMOTE_BKP_SERVER
       stripe on "MTA1:" at REMOTE_BKP_SERVER
   with init
   ```

   Dumps the transaction log for the *mydb* database, using the Backup Server REMOTE_BKP_SERVER. The Backup Server dumps approximately half the log to each of the two devices. The **init** option overwrites any existing files on the tape. On UNIX systems, the **retaindays** option specifies that the tapes cannot be overwritten for 14 days. (OpenVMS systems do not use **retaindays**; they always create new versions of dump files.)

3. **dump tran inventory_db to dev1 with standby_access**

   Dumps completed transactions from the *inventory_db* transaction log file to device *dev1*.

Comments

- Table 6-23 describes the commands and system procedures used to back up databases and logs.:

Table 6-23: Commands used to back up databases and logs

Use This Command	To Do This
dump database	Make routine dumps of the entire database, including the transaction log.
dump transaction	Make routine dumps of the transaction log, then truncate the inactive portion.
dump transaction with no_truncate	Dump the transaction log after failure of a database device.
dump transaction with truncate_only then **dump database**	Truncate the log without making a backup. Copy the entire database.
dump transaction with no_log then **dump database**	Truncate the log after your usual method fails due to insufficient log space. Copy the entire database.
sp_volchanged	Respond to the Backup Server's volume change messages.

Restrictions

- You cannot dump to the null device (on UNIX, */dev/null*; on OpenVMS, any device name beginning with "NL").
- You cannot use the **dump transaction** command in a transaction.
- When using 1/4-inch cartridge tape, you can dump only one database or transaction log per tape.
- You cannot issue dump the transaction log while the **trunc log on chkpt** database option is enabled or after enabling **select into/bulk copy/pllsort** and making minimally logged changes to the database with **select into**, fast bulk copy operations, default unlogged **writetext** operations, or a parallel sort. Use **dump database** instead.

◆ ***WARNING!***

Never modify the log table *syslogs* with a *delete*, *update*, or *insert* command.

- If a database does not have a log segment on a separate device from data segments, you cannot use **dump transaction** to copy the log and truncate it.
- If a user or threshold procedure issues a **dump transaction** command on a database where a **dump database** or another **dump transaction** is in progress, the second command sleeps until the first completes.
- To restore a database, use **load database** to load the most recent database dump; then use **load transaction** to load each subsequent transaction log dump **in the order in which it was made**.
- Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of these databases can cause database corruption.

- You cannot dump from an 11.x Adaptive Server to a 10.x Backup Server.
- You cannot have Sybase dumps and non-Sybase data (for example, UNIX archives) on the same tape.
- You cannot dump a transaction **with no_log** or **with truncate_only** if the database has offline pages.

Copying the Log After Device Failure: *with no_truncate*

- After device failure, use **dump transaction with no_truncate** to copy the log without truncating it. You can use this option only if your log is on a separate segment and your *master* database is accessible.
- The backup created by **dump transaction with no_truncate** is the most recent dump for your log. When restoring the database, load this dump last.

Databases Without Separate Log Segments: *with truncate_only*

- When a database does not have a log segment on a separate device from data segments, use **dump transaction with truncate_only** to remove committed transactions from the log without making a backup copy.

◆ ***WARNING!***

dump transaction with truncate_only provides no means to recover your databases. Run dump database at the earliest opportunity to ensure recoverability.

- Use **with truncate_only** on the *master*, *model*, and *sybsystemprocs* databases, which do not have log segments on a separate device from data segments.
- You can also use this option on very small databases that store the transaction log and data on the same device.
- Mission-critical user databases should have log segments on a separate device from data segments. Use the **log on** clause of **create database** to create a database with a separate log segment, or **alter database** and **sp_logdevice** to transfer the log to a separate device.

Dump Only Complete Transactions: *with standby_access*

- Use the **with standby_access** option to dump transaction logs for loading into a server that acts as a warm standby server for the database.
- When you use **with standby_access** to dump the transaction log, the dump proceeds to the furthest point in the log at which all earlier transactions have completed and there are no records belonging to open transactions.
- You must use **dump tran[saction]...with standby_access** in all situations where you will be loading two or more transaction logs in sequence and you want the database to be online between loads.
- After loading a dump made with the **with standby_access** option, use the **online database** command with the **for standby_access** option to make the database accessible.

◆ ***WARNING!***

If a transaction log contains open transactions and you dump it without the *with standby_access* option, version 11.9.2 does not allow you to load the log, bring the database online, then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after a load that was originally dumped with standby_access or after loading the entire series.

When All Else Fails: *with no_log*

- Use **dump transaction with no_log** only as a last resort, after your usual method of dumping the transaction log (**dump transaction** or **dump transaction with truncate_only**) fails because of insufficient log space.
- **dump transaction...with no_log** truncates the log without logging the dump transaction event. Because it copies no data, it requires only the name of the database.
- Every use of **dump transaction...with no_log** is considered an error and is recorded in Adaptive Server's error log.

◆ ***WARNING!***

dump transaction with no_log provides no means to recover your databases. Run dump database at the earliest opportunity to ensure recoverability.

- If you have created your databases with log segments on a separate device from data segments, written a last-chance threshold procedure that dumps your transaction log often enough, and allocated enough space to your log and database, you should not have to use this option. If you must use **with no_log**, increase the frequency of your dumps and the amount of log space.

Scheduling Dumps

- Transaction log dumps are **dynamic**—they can take place while the database is active. They may slow the system slightly, so run dumps when the database is not being heavily updated.
- Use **dump database** immediately after creating a database to make a copy of the entire database. You cannot run **dump transaction** on a new database until you have run **dump database**.
- Develop a regular schedule for backing up user databases and their transaction logs.
- **dump transaction** uses less storage space and takes less time than **dump database**. Typically, transaction log dumps are made more frequently than database dumps.

Using Thresholds to Automate *dump transaction*

- Use thresholds to automate backup procedures. To take advantage of Adaptive Server's last-chance threshold, create user

databases with log segments on a separate device from data segments.

- When space on the log segment falls below the last-chance threshold, Adaptive Server executes the last-chance threshold procedure. Including a **dump transaction** command in your last-chance threshold procedure helps protect you from running out of log space. For more information, see **sp_thresholdaction**.
- You can use **sp_addthreshold** to add a second threshold to monitor log space. For more information about thresholds, see the *System Administration Guide*.

Specifying Dump Devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You can specify a local dump device as:
  - A logical device name from the *sysdevices* system table
  - An absolute path name
  - A relative path name

  The Backup Server resolves relative path names using Adaptive Server's current working directory.
- Dumping to multiple stripes is supported for tape and disk devices. Placing multiple dumps on a device is supported only for tape devices.
- When dumping across the network, specify the absolute path name of the dump device. The path name must be valid on the machine on which the Backup Server is running. If the name includes any characters except letters, numbers, or the underscore (_), enclose it in quotes.
- Ownership and permissions problems on the dump device may interfere with use of **dump** commands. The **sp_addumpdevice** procedure adds the device to the system tables, but does not guarantee that you can dump to that device or create a file as a dump device.
- You can run more than one dump (or load) at the same time, as long as they use different dump devices.

Determining Tape Device Characteristics

- If you issue a **dump transaction** command without the **init** qualifier and Backup Server cannot determine the device type, the **dump transaction** command fails. For more information, see the *System Administration Guide.*

Backup Servers

- You must have a Backup Server running on the same machine as your Adaptive Server. (On OpenVMS systems, the Backup Server can be running in the same cluster as the Adaptive Server, as long as all database devices are visible to both.) The Backup Server must be listed in the *master..sysservers* table. This entry is created during installation or upgrade and should not be deleted.
- If your backup devices are located on another machine so that you dump across a network, you must also have a Backup Server installed on the remote machine.

Dump Files

- Dumping a log with the **init** option overwrites any existing files on the tape or disk.
- Dump file names identify which database was dumped and when the dump was made. If you do not specify a file name, Backup Server creates a default file name by concatenating the following:
  - Last seven characters of the database name
  - Two-digit year number
  - Three-digit day of the year (1– 366)
  - Hexadecimal-encoded time at which the dump file was created

For example, the file *cations930590E100* contains a copy of the *publications* database made on the fifty-ninth day of 1993:

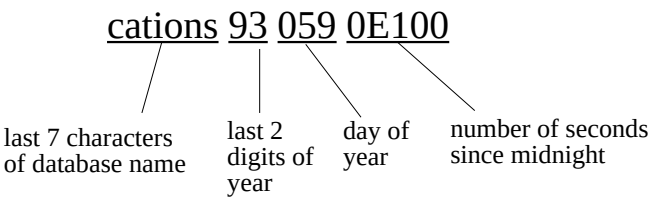

Figure 6-4: File naming convention for transaction log dumps

- The Backup Server sends the dump file name to the location specified by the **with notify** clause. Before storing a backup tape, the operator should label it with the database name, file name, date, and other pertinent information. When loading a tape without an identifying label, use the **with headeronly** and **with listonly** options to determine the contents.

Volume Names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.
- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

➤ ***Note***

When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing Dump Volumes

- On OpenVMS systems, the operating system requests a volume change when it detects the end of a volume or when the specified drive is offline. After mounting another volume, the operator uses the **REPLY** command to reply to these messages.
- On UNIX systems, the Backup Server requests a volume change when the tape capacity has been reached. After mounting another volume, the operator notifies the Backup Server by

executing the **sp_volchanged** system procedure on any Adaptive Server that can communicate with the Backup Server.

- If the Backup Server detects a problem with the currently mounted volume (for example, if the wrong volume is mounted), it requests a volume change by sending messages to either the client or its operator console. The operator responds to these messages with the **sp_volchanged** system procedure.

Appending to/Overwriting a Volume

- By default (**noinit**), Backup Server writes successive dumps to the same tape volume, making efficient use of high-capacity tape media. Data is added following the last end-of-tape mark. New dumps can be appended only to the last volume of a multivolume dump. Before writing to the tape, Backup Server verifies that the first file has not yet expired. If the tape contains non-Sybase data, Backup Server rejects it to avoid destroying potentially valuable information.
- Use the **init** option to reinitialize a volume. If you specify **init**, Backup Server overwrites any existing contents, even if the tape contains non-Sybase data, the first file has not yet expired, or the tape has ANSI access restrictions.
- Figure 6-5 illustrates how to dump three transaction logs to a single volume. Use:
  - **init** to initialize the tape for the first dump
  - **noinit** (the default) to append subsequent dumps
  - **unload** to rewind and unload the tape after the last dump

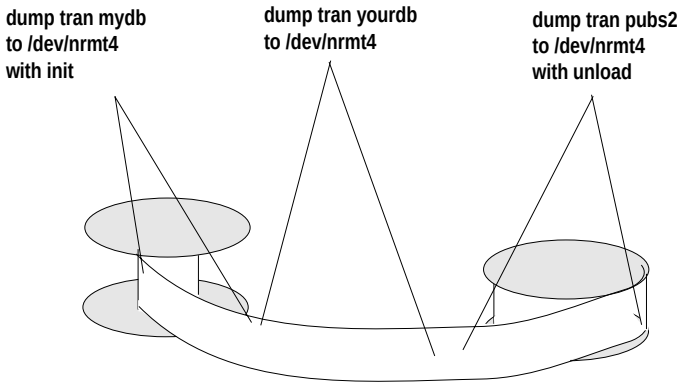

Figure 6-5: Dumping three transaction logs to a single volume

Dumping Logs Stored on Mirrored Devices

- At the beginning of a **dump transaction**, Adaptive Server passes the primary device name of each logical log device to the Backup Server. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If the named device fails before Backup Server completes its data transfer, Adaptive Server aborts the dump.
- If you attempt to unmirror a named log device while a **dump transaction** is in progress, Adaptive Server displays a message. The user executing the **disk unmirror** command can abort the dump or defer the **disk unmirror** until after the dump completes.
- **dump transaction with truncate_only** and **dump transaction with no_log** do not use the Backup Server. These commands are not affected when a log device is unmirrored, either by a device failure or by a **disk unmirror** command.
- **dump transaction** copies only the log segment. It is not affected when a data-only device is unmirrored, either by a device failure or by a **disk unmirror** command.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only System Administrators, users who have been granted the Operator role, and the Database Owner can execute **dump transaction**.

See Also

Commands	**dump database**, **load database**, **load transaction**, **online database**
System procedures	**sp_addumpdevice, sp_dboption, sp_dropdevice, sp_helpdevice, sp_logdevice, sp_volchanged**

execute

Function

Runs a procedure or dynamically executes Transact-SQL commands

Syntax

```
[exec[ute]] [@return_status = ]
   [[[server.]database.]owner.]procedure_name[;number]
       [[@parameter_name =] value |
           [@parameter_name =] @variable [output]
       [,[@parameter_name =] value |
           [@parameter_name =] @variable [output]...]]
   [with recompile]
```

or

```
exec[ute] ("string" | char_variable
   [+ "string" | char_variable]...)
```

Keywords and Options

execute | exec – is used to execute a stored procedure or an extended stored procedure (ESP). It is necessary only if the stored procedure call is **not** the first statement in a batch.

@return_status – is an optional integer variable that stores the return status of a stored procedure. It must be declared in the batch or stored procedure before it is used in an execute statement.

server – is the name of a remote server. You can execute a procedure on another Adaptive Server as long as you have permission to use that server and to execute the procedure in that database. If you specify a server name, but do not specify a database name, Adaptive Server looks for the procedure in your default database.

database – is the database name. Specify the database name if the procedure is in another database. The default value for *database* is the current database. You can execute a procedure in another database as long as you are its owner or have permission to execute it in that database.

owner – is the procedure owner's name. Specify the owner's name if more than one procedure of that name exists in the database. The default value for *owner* is the current user. The owner name is optional only if the Database Owner ("dbo") owns the procedure or if you own it.

procedure_name – is the name of a procedure defined with a **create procedure** statement.

;number – is an optional integer used to group procedures of the same name so that they can be dropped together with a single **drop procedure** statement. Procedures used in the same application are often grouped this way. For example, if the procedures used with an application named **orders** are named *orderproc;1*, *orderproc;2*, and so on, the statement:

```
drop proc orderproc
```

drops the entire group. Once procedures have been grouped, individual procedures within the group cannot be dropped. For example, you cannot execute the statement:

```
drop procedure orderproc;2
```

parameter_name – is the name of an argument to the procedure, as defined in the **create procedure** statement. Parameter names must be preceded by the @ sign.

If the "*@parameter_name = value*" form is used, parameter names and constants need not be supplied in the order defined in the **create procedure** statement. However, if this form is used for any parameter, it must be used for all subsequent parameters.

value – is the value of the parameter or argument to the procedure. If you do not use the "*@parameter_name = value*" form, you must supply parameter values in the order defined in the **create procedure** statement.

@variable – is the name of a variable used to store a return parameter.

output – indicates that the stored procedure is to return a return parameter. The matching parameter in the stored procedure must also have been created with the keyword **output**.

The **output** keyword can be abbreviated to **out**.

with recompile – forces compilation of a new plan. Use this option if the parameter you are supplying is atypical or if the data has significantly changed. The changed plan is used on subsequent executions. Adaptive Server ignores this option when executing an ESP.

string – is a literal string containing part of a Transact-SQL command to execute. There are no restrictions to the number of characters supplied with the literal string.

char_variable – is the name of a variable that supplies the text of a Transact-SQL command. The *char_variable* can supply a maximum of 255 characters.

Examples

1. ```
   execute showind titles
   ```

   or:

   ```
   exec showind @tabname = titles
   ```

   or, if this is the only statement in a batch or file:

   ```
   showind titles
   ```

   All three examples above execute the stored procedure *showind* with a parameter value *titles*.

2. ```
   declare @retstat int
   execute @retstat = GATEWAY.pubs.dbo.checkcontract
   "409-56-4008"
   ```

   Executes the stored procedure *checkcontract* on the remote server GATEWAY. Stores the return status indicating success or failure in *@retstat*.

3. ```
   declare @percent int
   select @percent = 10
   execute roy_check "BU1032", 1050, @pc = @percent
   output
   select Percent = @percent
   ```

   Executes the stored procedure *roy_check*, passing three parameters. The third parameter, *@pc*, is an **output** parameter. After execution of the procedure, the return value is available in the variable *@percent*.

4. ```
   create procedure
   showsysind @table varchar(30) = "sys%"
   as
     select sysobjects.name, sysindexes.name, indid
     from sysindexes, sysobjects
     where sysobjects.name like @table
     and sysobjects.id = sysindexes.id
   ```

   This procedure displays information about the system tables if the user does not supply a parameter.

5. ```
   declare @input varchar(12)
   select @input="Hello World!"
   execute xp_echo @in = @input, @out= @result output
   ```
   ```

Executes the extended stored procedure *xp_echo*, passing in a value of "Hello World!". The returned value of the extended stored procedure is stored in a variable named *result*.

6.
```
declare @tablename char(20)
declare @columname char(20)
select @tablename="sysobjects"
select @columname="name"
execute ('select ' + @columname + ' from ' +
@tablename + ' where id=3')
```

The final **execute** command concatenates string values and character variables to issue the Transact-SQL command:

```
select name from sysobjects where id=3
```

Comments

- Procedure results may vary, depending on the database in which they are executed. For example, the user-defined system procedure *sp_foo*, which executes the **db_name()** system function, returns the name of the database from which it is executed. When executed from the *pubs2* database, it returns the value "pubs2":

  ```
  exec pubs2..sp_foo

  ------------------------------
  pubs2

  (1 row affected, return status = 0)
  ```

  When executed from *sybsystemprocs*, it returns the value "sybsystemprocs":

  ```
  exec sybsystemprocs..sp_foo

  ------------------------------
  sybsystemprocs

  (1 row affected, return status = 0)
  ```

- There are two ways to supply parameters—by position, or by using:

  @parameter_name = value

  If you use the second form, you do not have to supply the parameters in the order defined in the **create procedure** statement.

  If you are using the **output** keyword and intend to use the return parameters in additional statements in your batch or procedure, the value of the parameter must be passed as a variable. For example:

  parameter_name = @variable_name

When executing an extended stored procedure, pass all parameters either by name or by value. You cannot mix parameters by value and parameters by name in a single invocation of the **execute** command for an ESP.

- You cannot use *text* and *image* columns as parameters to stored procedures or as values passed to parameters.
- It is an error to execute a procedure specifying **output** for a parameter that is not defined as a return parameter in the **create procedure** statement.
- You cannot pass constants to stored procedures using **output**; the return parameter requires a variable name. You must declare the variable's datatype and assign it a value before executing the procedure. Return parameters cannot have a datatype of *text* or *image*.
- It is not necessary to use the keyword **execute** if the statement is the first one in a batch. A batch is a segment of an input file terminated by the word "go" on a line by itself.
- Since the execution plan for a procedure is stored the first time it is run, subsequent run time is much shorter than for the equivalent set of standalone statements.
- Nesting occurs when one stored procedure calls another. The nesting level is incremented when the called procedure begins execution and it is decremented when the called procedure completes execution. Exceeding the maximum of 16 levels of nesting causes the transaction to fail. The current nesting level is stored in the *@@nestlevel* global variable.
- Return values 0 and -1 through -14 are currently used by Adaptive Server to indicate the execution status of stored procedures. Values from -15 through -99 are reserved for future use. See **return** for a list of values.
- Parameters are not part of transactions, so if a parameter is changed in a transaction which is later rolled back, its value does not revert to its previous value. The value that is returned to the caller is always the value at the time the procedure returns.
- If you use **select** * in your **create procedure** statement, the procedure does not pick up any new columns you may have added to the table (even if you use the **with recompile** option to execute). You must **drop** the procedure and re-create it.
- Commands executed via remote procedure calls cannot be rolled back.

- The **with recompile** option is ignored when Adaptive Server executes an extended stored procedure.

Dynamically Executing Transact-SQL

- When used with the *string* or *char_variable* options, **execute** concatenates the supplied strings and variables to execute the resulting Transact-SQL command. This form of the **execute** command may be used in SQL batches, procedures, and triggers.
- You cannot supply *string* and *char_variable* options to execute the following commands: **begin transaction**, **commit**, **declare cursor**, **rollback**, **dump transaction**, **dbcc**, **set**, **use**, or nested **execute** commands.
- The **create view** command can be specified using **execute()**, but only in SQL batches. **create view** cannot be used in procedures, either as a static command or as a string parameter to **execute()**.
- The contents of the *string* or *char_variable* options cannot reference local variables declared in the SQL batch or procedure
- *string* and *char_variable* options can be concatenated to create new tables. Within the same SQL batch or procedure, however, the table created with **execute()** is visible only to other **execute()** commands. After the SQL batch or procedure has completed, the dynamically-created table is persistent and visible to other commands.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

execute permission defaults to the owner of the procedure, who can transfer it to other users.

The permission to execute Transact-SQL commands defined with the *string* or *char_variable* options is checked against the user executing the command. This is true even when **execute()** is defined within a procedure or trigger that belongs to another user.

See Also

Commands	**create procedure, drop procedure, return**
System procedures	**sp_addextendedproc, sp_depends, sp_dropextendedproc, sp_helptext**

fetch

Function

Returns a row or a set of rows from a cursor result set.

Syntax

```
fetch cursor_name [ into fetch_target_list ]
```

Parameters

cursor_name – the name of the cursor

into *fetch_target_list* – is a comma-separated list of parameters or local variables into which cursor results are placed. The parameters and variables must be declared prior to the **fetch**.

Examples

```
1. fetch authors_crsr
```

Returns a row of information from the cursor result set defined by the *authors_crsr* cursor.

```
2. fetch pubs_crsr into @name, @city, @state
```

Returns a row of information from the cursor result set defined by the *pubs_crsr* cursor into the variables *@name*, *@city*, and *@state*.

Comments

Restrictions

- Before you can use **fetch**, you must declare the cursor and **open** it.
- The *cursor_name* cannot be a Transact-SQL parameter or local variable.
- You cannot **fetch** a row that has already been fetched. There is no way to backtrack through the result set, but you can close and reopen the cursor to create the cursor result set again and start from the beginning.
- Adaptive Server expects a one-to-one correspondence between the variables in the *fetch_target_list* and the target list expressions specified by the *select_statement* that defines the cursor. The datatypes of the variables or parameters must be compatible with the datatypes of the columns in the cursor result set.

- When you set chained transaction mode, Adaptive Server implicitly begins a transaction with the **fetch** statement if no transaction is currently active. However, this situation occurs only when you set the **close on endtran** option and the cursor remains open after the end of the transaction that initially opened it, since the **open** statement also automatically begins a transaction.

Cursor Position

- After you **fetch** all the rows, the cursor points to the last row of the result set. If you **fetch** again, Adaptive Server returns a warning through the *@@sqlstatus* variable indicating there is no more data, and the cursor position moves beyond the end of the result set. You can no longer **update** or **delete** from that current cursor position.
- With **fetch into**, Adaptive Server does not advance the cursor position when an error occurs because the number of variables in the *fetch_target_list* does not equal the number of target list expressions specified by the query that defines the cursor. However, it does advance the cursor position, even if a compatibility error occurs between the datatypes of the variables and the datatypes of the columns in the cursor result set.

Determining How Many Rows Are Fetched

- You can **fetch** one or more rows at a time. Use the **cursor rows** option of the **set** command to specify the number of rows to **fetch**.

Getting Information About Fetches

- The *@@sqlstatus* global variable holds status information (warning exceptions) resulting from the execution of a **fetch** statement. The value of *@@sqlstatus* is 0, 1, or 2, as shown in Table 6-24.

Table 6-24: @@sqlstatus values

0	Indicates successful completion of the **fetch** statement.
1	Indicates that the **fetch** statement resulted in an error.
2	Indicates that there is no more data in the result set. This warning can occur if the current cursor position is on the last row in the result set and the client submits a **fetch** statement for that cursor.

Only a **fetch** statement can set *@@sqlstatus*. Other statements have no effect on *@@sqlstatus*.

- The *@@rowcount* global variable holds the number of rows returned from the cursor result set to the client up to the last **fetch**. In other words, it represents the total number of rows seen by the client at any one time.

  Once all the rows have been read from the cursor result set, *@@rowcount* represents the total number of rows in the cursor results set. Each open cursor is associated with a specific *@@rowcount* variable, which is dropped when you close the cursor. Check *@@rowcount* after a **fetch** to get the number of rows read for the cursor specified in that **fetch**.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of variables in a target list and fetch of multiple rows are Transact-SQL extensions.

Permissions

fetch permission defaults to all users.

See Also

Commands	declare cursor, open, set

goto Label

Function

Branches to a user-defined label.

Syntax

```
label:
   goto label
```

Examples

```
1. declare @count smallint
   select @count = 1
   restart:
     print "yes"
   select @count = @count + 1
   while @count <=4
     goto restart
```

Shows the use of a label called *restart*.

Comments

- The label name must conform to the rules for identifiers and must be followed by a colon (:) when it is declared. It is not followed by a colon when it is used with **goto**.
- Make the **goto** dependent on an **if** or **while** test, or some other condition, to avoid an endless loop between **goto** and the label.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

goto permission defaults to all users. No permission is required to use it.

See Also

Commands	if...else, while

grant

Function

Assigns permissions to users or to user-defined roles. Assigns roles to users or system or user-defined roles.

Syntax

To grant permission to access database objects:

```
grant {all [privileges]| permission_list}
   on { table_name [(column_list)]
       | view_name[(column_list)]
       | stored_procedure_name}
   to {public | name_list | role_name}
   [with grant option]
```

To grant permission to execute certain commands:

```
grant {all [privileges] | command_list}
   to {public | name_list | role_name}
```

To grant a role to a user or a role:

```
grant {role role_granted [, role_granted ...]}
   to grantee [, grantee...]
```

Keywords and Options

all – when used to assign permission to access database objects (the first syntax format), **all** specifies that all permissions applicable to the specified object are granted. All object owners can use **grant all** with an object name to grant permissions on their own objects.

Only a System Administrator or the Database Owner can assign permission to create database objects (the second syntax format). When used by a System Administrator, **grant all** assigns all **create** permissions (**create database**, **create default**, **create procedure**, **create rule**, **create table**, and **create view**). When the Database Owner uses **grant all**, Adaptive Server grants all **create** permissions except **create database**, and prints an informational message.

Specifying **all** does not include permission to execute **set proxy** or **set session authorization**.

permission_list – is a list of object access permissions granted. If more than one permission is listed, separate them with commas. The

following table illustrates the access permissions that can be granted on each type of object:

Object	*permission_list* Can Include:
Table	**select, insert, delete, update, references**
View	**select, insert, delete, update**
Column	**select, update, references** Column names can be specified in either *permission_list* or *column_list* (see example 2).
Stored procedure	**execute**

command_list – is a list of commands that the user can execute. If more than one command is listed, separate them with commas. The command list can include **create database**, **create default**, **create procedure**, **create rule**, **create table**, **create view**, **set proxy**, and **set session authorization**.

create database permission can be granted only by a System Administrator, and only from within the *master* database.

Only a System Security Officer can grant users permission to execute **set proxy** or **set session authorization**. Granting permission to execute **set proxy** or **set session authorization** allows the grantee to impersonate another login in the server. **set proxy** and **set session authorization** are identical, except that **set session authorization** follows the ANSI92 standard, and **set proxy** is a Transact-SQL extension.

table_name – is the name of the table on which you are granting permissions. The table must be in your current database. Only one object can be listed for each **grant** statement.

column_list – is a list of columns, separated by commas, to which the permissions apply. If columns are specified, only **select**, **references**, and **update** permissions can be granted.

view_name – is the name of the view on which you are granting permissions. The view must be in your current database. Only one object can be listed for each **grant** statement.

stored_procedure_name – is the name of the stored procedure on which you are granting permissions. The stored procedure must be in your current database. Only one object can be listed for each **grant** statement.

public – is all users. For object access permissions, **public** excludes the object owner. For object creation permissions or **set proxy** authorizations, **public** excludes the Database Owner. You cannot **grant** permissions **with grant option** to "public" or to other groups or roles.

name_list – is a list of users' database names and / or group names, separated by commas.

with grant option – allows the users specified in *name_list* to grant object access permissions to other users. You can grant permissions **with grant option** only to individual users, not to "public" or to a group or role.

role – grants a role to a user or to a system or user-defined role.

role_granted – is the name of a system or user-defined role that the System Security Officer is granting to a user or a role.

grantee – is the name of a system role, user-defined role, or a user, to whom you are granting a role.

role_name – is the name of a system or user-defined role to which you are granting the permission.

Examples

1. ```
   grant insert, delete
   on titles
   to mary, sales
   ```

   Grants Mary and the "sales" group permission to use the **insert** and **delete** commands on the *titles* table.

2. ```
   grant update
   on titles (price, advance)
   to public
   ```

   or:

   ```
   grant update (price, advance)
   on titles
   to public
   ```

   Two ways to grant **update** permission on the *price* and *advance* columns of the *titles* table to "public" (which includes all users).

3. ```
   grant set proxy to harry, billy
   ```

   Grants Harry and Billy permission to execute either **set proxy** or **set session authorization** to impersonate another user in the server.
   ```

4. **grant set session authorization to sso_role**

   Grants users with **sso_role** permission to execute either **set proxy** or **set session authorization** to impersonate another user in the server.

5. **grant set proxy to vip_role**

   Grants users with **vip_role** the ability to impersonate another user in the server. **vip_role** must be a role defined by a System Security Officer with the **create role** command.

6. **grant create database, create table**
   to mary, john

   Grants Mary and John permission to use the **create database** and **create table** commands. Because **create database** permission is being granted, this command can be executed only by a System Administrator within the *master* database. Mary and John's **create table** permission applies only to the *master* database.

7. **grant all on titles**
   to public

   Grants complete access permissions on the *titles* table to all users.

8. **grant all**
   to public

   Grants all object creation permissions in the current database to all users. If this command is executed by a System Administrator from the *master* database, it includes **create database** permission.

9. **grant update on authors**
   to mary
   with grant option

   Gives Mary permission to use the **update** command on the *authors* table and to grant that permission to others.

10. **grant select, update on titles(price)**
    to bob
    with grant option

    Gives Bob permission to use the **select** and **update** commands on the *price* column of the *titles* table and to grant that permission to others.

11. **grant execute on new_sproc**
    to sso_role

    Grants permission to execute the *new_sproc* stored procedure to all System Security Officers.

```
12.grant references on titles(price)
   to james
```

Grants James permission to create a referential integrity constraint on another table that refers to the *price* column of the *titles* table.

```
13.grant role specialist_role to doctor_role
```

Grants the role “specialist”, with all its permissions and privileges, to the role “doctor”.

```
14.grant role doctor_role to mary
```

Grants the role “doctor” to Mary.

Comments

- You can substitute the word **from** for **to** in the **grant** syntax.
- Table 6-25 summarizes default permissions on Transact-SQL commands in Adaptive Server. The user listed under the “Defaults To” heading is the lowest level of user that is automatically granted permission to execute a command. This user can **grant** or **revoke** the permission if it is transferable. Users at higher levels than the default are either automatically assigned permission or (in the case of Database Owners) can get permission by using the **setuser** command.

  For example, the owner of a database does not automatically receive permission on objects owned by other users. A Database Owner can gain such permission by assuming the identity of the object owner with the **setuser** command, and then issuing the appropriate **grant** or **revoke** statement. System Administrators have permission to access all commands and objects at any time.

  The Adaptive Server installation script assigns a set of permissions to the default group “public.” **grant** and **revoke** statements need not be written for these permissions.

Table 6-25 does not include the System Security Officer, who does not have any special permissions on commands and objects, but only on certain system procedures.

Table 6-25: Command and object permissions

Statement	Defaults To					Can Be Granted/Revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	N/A
alter database			•			(1)		
alter role								•
alter table				•			•	
begin transaction					•			•
checkpoint			•				•	
commit					•			•
create database	•					•		
create default			•			•		
create index				•			•	
create procedure			•			•		
create role								•
create rule			•			•		
create table			•		(2)	• (2)		
create trigger					•	•		
create view			•			•		
dbcc	Varies depending upon options. See **dbcc** in this manual.						•	
delete				• (3)		•		
disk init	•						•	
disk mirror	•							
disk refit	•							

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner

(5) Transferred with **select** permission
(6) Transferred with **update** permission
"No" means use of the command is never restricted
"N/A" means use of the command is always restricted

Table 6-25: Command and object permissions (continued)

Statement	Defaults To					Can Be Granted/Revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	N/A
disk reinit	•							
disk remirror	•							
disk unmirror	•						•	
drop (any object)				•			•	
dump database		•	•				•	
dump transaction		•	•				•	
execute				•(4)		•		
grant on object				•		•		
grant command			•			•		
insert				• (3)		•		
kill	•						•	
load database		•	•				•	
load transaction		•	•				•	
print					•			•
raiserror					•			•
readtext				•		(5)		
revoke on object				•			•	
revoke command			•				•	
rollback					•			•
save transaction					•			•
select				• (3)		•		
set					•			•
setuser			•				•	

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner

(5) Transferred with **select** permission
(6) Transferred with **update** permission
"No" means use of the command is never restricted
"N/A" means use of the command is always restricted

Table 6-25: Command and object permissions (continued)

Statement	Defaults To					Can Be Granted/Revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	N/A
shutdown	•						•	
truncate table				•			•	
update				• (3)		•		
update all statistics				•			•	
update partition statistics				•			•	
update statistics				•			•	
writetext				•		(6)		

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner
(5) Transferred with **select** permission
(6) Transferred with **update** permission
"No" means use of the command is never restricted
"N / A" means use of the command is always restricted

- You can grant permissions only on objects in your current database.
- Before you create a table that includes a referential integrity constraint to reference another user's table, you must be granted **references** permission on that referenced table (see example 10). The table must also include a unique constraint or unique index on the referenced columns. See **create table** for more information about referential integrity constraints.
- **grant** and **revoke** commands are order-sensitive. The command that takes effect when there is a conflict is the one issued most recently.
- A user can be granted permission on a view or stored procedure even if he or she has no permissions on objects referenced by the procedure or view. For more information, see the *System Administration Guide.*
- Adaptive Server grants all users permission to declare cursors, regardless of the permissions defined for the base tables or views referenced in the **declare cursor** statement. Cursors are not defined as Adaptive Server objects (such as tables), so no permissions can be applied against a cursor. When a user opens a cursor, Adaptive

Server determines whether the user has **select** permissions on the objects that define that cursor's result set. It checks permissions each time a cursor is opened.

If the user has permission to access the objects defined by the cursor, Adaptive Server opens the cursor and allows the user to **fetch** row data through the cursor. Adaptive Server does not apply permission checking for each **fetch**. However, if the user performs a **delete** or an **update** through that cursor, the regular permission checking applies for deleting and updating the data of objects referenced in the cursor result set.

- A **grant** statement adds one row to the *sysprotects* system table for each user, group, or role that receives the permission. If you subsequently **revoke** the permission from the user or group, Adaptive Server removes the row from *sysprotects*. If you revoke the permission from selected group members only, but not from the entire group to which it was granted, Adaptive Server retains the original row and adds a new row for the revoke.
- If a user inherits a particular permission by virtue of being a member of a group, and the same permission is explicitly granted to the user, no row is added to *sysprotects*. For example, if "public" has been granted **select** permission on the *phone* column in the *authors* table, then John, a member of "public," is granted **select** permission on all columns of *authors*. The row added to *sysprotects* as a result of the **grant** to John will contain references to all columns in the *authors* table except for the *phone* column, on which he already had permission.
- Permission to issue the **create trigger** command is granted to users by default. When you revoke permission for a user to create triggers, a revoke row is added in the *sysprotects* table for that user. To grant permission to that user to issue **create trigger**, you must issue two **grant** commands. The first command removes the revoke row from *sysprotects*; the second inserts a grant row. If you revoke permission to create triggers, the user cannot create triggers even on tables that the user owns. Revoking permission to create triggers from a user affects only the database where the revoke command was issued.
- You can get information about permissions with these system procedures:
  - **sp_helprotect** reports permissions information for a database object or a user.

- **sp_column_privileges** reports permissions information for one or more columns in a table or view.
- **sp_table_privileges** reports permissions information for all columns in a table or view.
- **sp_activeroles** displays all active roles for the current login session of Adaptive Server.
- **sp_displayroles** displays all roles granted to another role, or displays the entire hierarchy tree of roles in table format.

grant all (Object Creation Permissions)

- When used with only user or group names (no object names), **grant all** assigns these permissions: **create database**, **create default**, **create procedure**, **create rule**, **create table**, and **create view**. **create database** permission can be granted only by a System Administrator and only from within the *master* database.
- Only the Database Owner and a System Administrator can use the **grant all** syntax without an object name to grant **create** command permissions to users or groups. When the **grant all** command is used by the Database Owner, an informational message is printed, stating that only a System Administrator can grant **create database** permission. All other permissions noted above are granted.
- All object owners can use **grant all** with an object name to grant permissions on their own objects. When used with a table or view name plus user or group names, **grant all** enables **delete**, **insert**, **select**, and **update** permissions on the table.

grant with grant option Rules

- You cannot grant permissions **with grant option** to "public" or to a group or role.
- In granting permissions, a System Administrator is treated as the object owner. If a System Administrator grants permission on another user's object, the owner's name appears as the grantor in *sysprotects* and in **sp_helprotect** output.
- Information for each **grant** is kept in the system table *sysprotects* with the following exceptions:
  - Adaptive Server displays an informational message if a specific permission is granted to a user more than once by the same grantor. Only the first **grant** is kept.

- If two **grant**s are exactly same except that one of them is granted **with grant option**, the **grant with grant option** is kept.
- If two **grant** statements grant the same permissions on a particular table to a specific user, but the columns specified in the grants are different, Adaptive Server treats the grants as if they were one statement. For example, the following **grant** statements are equivalent:

```
grant select on titles(price, contract) to keiko
grant select on titles(advance) to keiko

grant select on titles(price, contract, advance)
to keiko
```

Granting Proxies and Session Authorizations

- Granting permission to execute **set proxy** or **set session authorization** allows the grantee to impersonate another login in Adaptive Server. **set proxy** and **set session authorization** are identical with one exception: **set session authorization** follows the SQL standard, and **set proxy** is a Transact-SQL extension.
- To grant **set proxy** or **set session authorization** permission, you must be a System Security Officer, and you must be in the *master* database.
- The name you specify in the **grant set proxy** command must be a valid user in the database; that is, the name must be in the *sysusers* table in the database.
- **grant all** does **not** include the **set proxy** or **set session authorization** permissions.

Granting Permission to Roles

- You can use the **grant** command to grant permissions to all users who have been granted a specified role. The role can be either a system role, like **sso_role** or **sa_role**, or a user-defined role. For a user-defined role, the System Security Officer must create the role with a **create role** command.

  However, **grant execute** permission does not prevent users who do not have a specified role from being individually granted permission to execute a stored procedure. If you want to ensure, for example, that only System Security Officers can ever be granted permission to execute a stored procedure, use the **proc_role** system function within the stored procedure itself. It checks to see whether the invoking user has the correct role to execute the procedure. See **proc_role** for more information.

- Permissions that are granted to roles override permissions that are granted to users or groups. For example, say John has been granted the System Security Officer role, and **sso_role** has been granted permission on the *sales* table. If John's individual permission on *sales* is revoked, he can still access *sales* because his role permissions override his individual permissions.

Users and User Groups

- User groups allow you to **grant** or **revoke** permissions to more than one user with a single statement. Each user can be a member of only one group and is always a member of "public."
- The Database Owner or System Administrator can add new users with **sp_adduser** and create groups with **sp_addgroup**. To allow users with logins on Adaptive Server to use the database with limited privileges, you can add a "guest" user with **sp_adduser** and assign limited permissions to "guest". All users with logins can access the database as "guest".
- To remove a user, use **sp_dropuser**. To remove a group, use **sp_dropgroup**.

  To add a new user to a group other than "public," use **sp_adduser**. To change an established user's group, use **sp_changegroup**.

  To display the members of a group, use **sp_helpgroup**.
- When **sp_changegroup** is executed to change group membership, it clears the in-memory protection cache by executing:

  ```
  grant all to null
  ```

  so that the cache can be refreshed with updated information from the *sysprotects* table. If you need to modify *sysprotects* directly, contact Sybase Technical Support.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	Granting permissions to groups and granting **set proxy** are Transact-SQL extensions. Granting **set session authorization** (identical in function to **set proxy**) follows the ANSI standard.

Permissions

Database Object Access Permissions

grant permission for database objects defaults to object owners. An object owner can grant permission to other users on his or her own database objects.

Command Execution Permissions

Only a System Administrator can grant **create database** permission, and only from the *master* database. Only a System Security Officer can grant **create trigger** permission.

Proxy and Session Authorization Permissions

Only a System Security Officer can grant **set proxy** or **set session authorization**, and only from the *master* database.

Role Permissions

You can grant roles only from the *master* database. Only a System Security Officer can grant **sso_role**, **oper_role** or a user-defined role to a user or a role. Only System Administrators can grant **sa_role** to a user or a role. Only a user who has both **sa_role** and **sso_role** can grant a role which includes **sa_role**.

See Also

Catalog stored procedures	sp_column_privileges
Commands	revoke, setuser, set
Functions	proc_role
System procedures	sp_addgroup, sp_adduser, sp_changedbowner, sp_changegroup, sp_dropgroup, sp_dropuser, sp_helpgroup, sp_helprotect, sp_helpuser, sp_role

group by and having Clauses

Function

Used in **select** statements to divide a table into groups and to return only groups that match conditions in the **having** clause.

Syntax

```
Start of select statement
[group by [all] aggregate_free_expression
   [, aggregate_free_expression]...]
[having search_conditions]
End of select statement
```

Keywords and Options

group by – specifies the groups into which the table will be divided, and if aggregate functions are included in the select list, finds a summary value for each group. These summary values appear as columns in the results, one for each group. You can refer to these summary columns in the **having** clause.

You can use the **avg**, **count**, **max**, **min**, and **sum** aggregate functions in the select list before **group by** (the expression is usually a column name). For more information, see "Aggregate Functions" in Chapter 2, "Transact-SQL Functions".

A table can be grouped by any combination of columns—that is, groups can be nested within each other, as in example 2.

all – is a Transact-SQL extension that includes all groups in the results, even those excluded by a **where** clause. For example:

```
select type, avg(price)
from titles
where advance > 7000
group by all type
```

```
type
-----------------  ----------
UNDECIDED                NULL
business                 2.99
mod_cook                 2.99
popular_comp            20.00
psychology               NULL
trad_cook               14.99

(6 rows affected)
```

"NULL" in the aggregate column indicates groups that would be excluded by the **where** clause. A **having** clause negates the meaning of **all**.

aggregate_free_expression – is an expression that includes no aggregates. A Transact-SQL extension allows grouping by an aggregate-free expression as well as by a column name.

You cannot group by column heading or alias. This example is correct:

```
select Price=avg(price), Pay=avg(advance),
Total=price * $1.15
from titles
group by price * $1.15
```

having – sets conditions for the **group by** clause, similar to the way in which **where** sets conditions for the **select** clause.

having search conditions can include aggregate expressions; otherwise, **having** search conditions are identical to **where** search conditions. Following is an example of a **having** clause with aggregates:

```
select pub_id, total = sum(total_sales)
from titles
where total_sales is not null
group by pub_id
having count(*)>5
```

When Adaptive Server optimizes queries, it evaluates the search conditions in **where** and **having** clauses, and determines which conditions are search arguments (SARGs) that can be used to choose the best indexes and query plan. For each table in a query, a maximum of 128 search arguments can be used to optimize the query. All of the search conditions, however, are used to qualify the rows. For more information on search arguments, see the *Performance and Tuning Guide*.

Examples

1.
```
select type, avg(advance), sum(total_sales)
from titles
group by type
```

   Calculates the average advance and the sum of the sales for each type of book.

2.
```
select type, pub_id, avg(advance), sum(total_sales)
from titles
group by type, pub_id
```

   Groups the results by type, then by *pub_id* within each type.

3.
```
select type, avg(price)
from titles
group by type
having type like 'p%'
```

   Calculates results for all groups, but displays only groups whose type begins with "p".

4.
```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > $15000
and avg(price) < $10
and pub_id > "0700"
```

   Calculates results for all groups, but displays results for groups matching the multiple conditions in the **having** clause.

5.
```
select p.pub_id, sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id
```

   Calculates the total sales for each group (publisher) after joining the *titles* and *publishers* tables.

6.
```
select title_id, advance, price
from titles
where advance > 1000
having price > avg(price)
```

   Displays the titles that have an advance of more than $1000 and a price that is more than the average price of all titles.

Comments

- You can use a column name or any expression (except a column heading or alias) after **group by.** You can use **group by** to calculate results or display a column or an expression that does not appear

in the select list (a Transact-SQL extension described in "Transact-SQL Extensions to group by and having").

- The maximum number of columns or expressions allowed in a **group by** clause is 16.
- The sum of the maximum lengths of all the columns specified by the **group by** clause cannot exceed 256 bytes.
- Null values in the **group by** column are put into a single group.
- You cannot name *text* or *image* columns in **group by** and **having** clauses.
- You cannot use a **group by** clause in the **select** statement of an updatable cursor.
- Aggregate functions can be used only in the select list or in a **having** clause. They cannot be used in a **where** or **group by** clause.

  Aggregate functions are of two types. Aggregates applied to **all the qualifying rows in a table** (producing a single value for the whole table per function) are called **scalar aggregates**. An aggregate function in the select list with no **group by** clause applies to the whole table; it is one example of a scalar aggregate.

  Aggregates applied to **a group of rows in a specified column or expression** (producing a value for each group per function) are called **vector aggregates**. For either aggregate type, the results of the aggregate operations are shown as new columns that the **having** clause can refer to.

  You can nest a vector aggregate inside a scalar aggregate. See "Aggregate Functions" in Chapter 2, "Transact-SQL Functions" for more information.

How *group by* and *having* Queries with Aggregates Work

- The **where** clause excludes rows that do not meet its search conditions; its function remains the same for grouped or non-grouped queries.
- The **group by** clause collects the remaining rows into one group for each unique value in the **group by** expression. Omitting **group by** creates a single group for the whole table.
- Aggregate functions specified in the select list calculate summary values for each group. For scalar aggregates, there is only one value for the table. Vector aggregates calculate values for the distinct groups.

- The **having** clause excludes groups from the results that do not meet its search conditions. Even though the **having** clause tests only rows, the presence or absence of a **group by** clause may make it appear to be operating on groups:
  - When the query includes **group by**, **having** excludes result group rows. This is why **having** seems to operate on groups.
  - When the query has no **group by**, **having** excludes result rows from the (single-group) table. This is why **having** seems to operate on rows (the results are similar to **where** clause results).

Standard *group by* and *having* Queries

- All **group by** and **having** queries in the "Examples" section adhere to the SQL standard. It dictates that queries using **group by**, **having**, and vector aggregate functions produce one row and one summary value per group, using these guidelines:
  - Columns in a select list must also be in the **group by** expression, or they must be arguments of aggregate functions.
  - A **group by** expression can contain only column names that are in the select list. However, columns used only as arguments of aggregate functions in the select list do not qualify.
  - Columns in a **having** expression must be single-valued—arguments of aggregates, for instance—and they must be in the select list or **group by** clause. Queries with a select list aggregate and a **having** clause **must** have a **group by** clause. If you omit the **group by** for a query without a select list aggregate, all the rows not excluded by the **where** clause are considered to be a single group (see example 6).

  In non-grouped queries, the principle that "**where** excludes rows" seems straightforward. In grouped queries, the principle expands to "**where** excludes rows before **group by**, and **having** excludes rows from the display of results."
- The SQL standard allows queries that join two or more tables to use **group by** and **having**, if they also adhere to the above guidelines. When specifying joins or other complex queries, use the standard syntax of **group by** and **having** until you fully comprehend the effect of the Transact-SQL extensions to both clauses, as described in "Transact-SQL Extensions to group by and having."

  To help you avoid problems with extensions, Adaptive Server provides the **fipsflagger** option to the **set** command that issues a

non-fatal warning for each occurrence of a Transact-SQL extension in a query. See **set** for more information.

Transact-SQL Extensions to *group by* and *having*

- Transact-SQL extensions to standard SQL make displaying data more flexible, by allowing references to columns and expressions that are not used for creating groups or summary calculations:
  - A select list that includes aggregates can include **extended** columns that are not arguments of aggregate functions and are not included in the **group by** clause. An extended column affects the display of final results, since additional rows are displayed.
  - The **group by** clause can include columns or expressions that are not in the select list.
  - The **group by all** clause displays all groups, even those excluded from calculations by a **where** clause. See the example for the keyword **all** in the "Keywords and Options" section.
  - The **having** clause can include columns or expressions that are not in the select list and not in the **group by** clause.

  When the Transact-SQL extensions add rows and columns to a display, or if **group by** is omitted, query results can be hard to interpret. The examples that follow can help you understand how Transact-SQL extensions can affect query results.
- The following examples illustrate the differences between queries that use standard **group by** and **having** clauses and queries that use the Transact-SQL extensions:

1.
```
select type, avg(price)
from titles
group by type
```

```
type
----------------------  ----------
UNDECIDED                     NULL
business                     13.73
mod_cook                     11.49
popular_comp                 21.48
psychology                   13.50
trad_cook                    15.96

(6 rows affected)
```

An example of a standard grouping query.

```
2. select type, price, avg(price)
   from titles
   group by type
```

```
type         price
------------ --------------- ----------------
business               19.99            13.73
business               11.95            13.73
business                2.99            13.73
business               19.99            13.73
mod_cook               19.99            11.49
mod_cook                2.99            11.49
UNDECIDED               NULL             NULL
popular_comp           22.95            21.48
popular_comp           20.00            21.48
popular_comp            NULL            21.48
psychology             21.59            13.50
psychology             10.95            13.50
psychology              7.00            13.50
psychology             19.99            13.50
psychology              7.99            13.50
trad_cook              20.95            15.96
trad_cook              11.95            15.96
trad_cook              14.99            15.96

(18 rows affected)
```

The Transact-SQL extended column, *price* (in the select list, but not an aggregate and not in the **group by** clause), causes all qualified rows to display in each qualified group, even though a standard **group by** clause produces a single row per group. The **group by** still affects the vector aggregate, which computes the average price per group displayed on each row of each group (they are the same values that were computed for example 1).

```
3. select type, price, avg(price)
   from titles
   where price > 10.00
   group by type
```

```
type         price
------------ --------------- ----------------
business               19.99            17.31
business               11.95            17.31
business                2.99            17.31
business               19.99            17.31
mod_cook               19.99            19.99
mod_cook                2.99            19.99
popular_comp           22.95            21.48
popular_comp           20.00            21.48
popular_comp            NULL            21.48
psychology             21.59            17.51
psychology             10.95            17.51
psychology              7.00            17.51
psychology             19.99            17.51
psychology              7.99            17.51
trad_cook              20.95            15.96
trad_cook              11.95            15.96
trad_cook              14.99            15.96

(17 rows affected)
```

The way Transact-SQL extended columns are handled can make it look as if a query is ignoring a **where** clause. This query computes the average prices using only those rows that satisfy the **where** clause, but it also displays rows that do not match the **where** clause.

Adaptive Server first builds a worktable containing only the type and aggregate values using the **where** clause. This worktable is joined back to the *titles* table in the grouping column *type* to include the *price* column in the results, but the **where** clause is **not** used in the join.

The only row in *titles* that is not in the results is the lone row with *type* = "UNDECIDED" and a NULL price, that is, a row for which there were no results in the worktable. If you also want to eliminate the rows from the displayed results that have prices of less than $10.00, you must add a **having** clause that repeats the **where** clause, as shown in example 4.

```
4. select type, price, avg(price)
   from titles
   where price > 10.00
   group by type
   having price > 10.00
```

```
type           price
-----------  ---------------  ----------------
business                19.99           17.31
business                11.95           17.31
business                19.99           17.31
mod_cook                19.99           19.99
popular_comp            22.95           21.48
popular_comp            20.00           21.48
psychology              21.59           17.51
psychology              10.95           17.51
psychology              19.99           17.51
trad_cook               20.95           15.96
trad_cook               11.95           15.96
trad_cook               14.99           15.96

(12 rows affected)
```

If you are specifying additional conditions, such as aggregates, in the **having** clause, be sure to also include all conditions specified in the **where** clause. Adaptive Server will appear to ignore any **where** clause conditions that are missing from the **having** clause.

```
5. select p.pub_id, t.type, sum(t.total_sales)
   from publishers p, titles t
   where p.pub_id = t.pub_id
   group by p.pub_id, t.type
```

```
pub_id  type
------  ------------  --------
0736    business         18722
0736    psychology        9564
0877    UNDECIDED         NULL
0877    mod_cook         24278
0877    psychology         375
0877    trad_cook        19566
1389    business         12066
1389    popular_comp     12875

(8 rows affected)
```

This is an example of a standard grouping query using a join between two tables. It groups by *pub_id*, then by *type* within each publisher ID, to calculate the vector aggregate for each row. It

may seem that it is only necessary to specify **group by** for the *pub_id* and *type* columns to produce the results, and add extended columns as follows:

```
select p.pub_id, p.pub_name, t.type,
    sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id, t.type
```

However, the results for the above query are much different from the results for the first query in this example. After joining the two tables to determine the vector aggregate in a worktable, Adaptive Server joins the worktable to the table (*publishers*) of the extended column for the final results. Each extended column from a different table invokes an additional join.

As you can see, using the extended column extension in queries that join tables can easily produce results that are difficult to comprehend. In most cases, you should use the standard **group by** to join tables in your queries.

6.
```
select p.pub_id, sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id, t.type
```

```
pub_id
------  --------
0736       18722
0736        9564
0877        NULL
0877       24278
0877         375
0877       19566
1389       12066
1389       12875

(8 rows affected)
```

This example uses the Transact-SQL extension to **group by** to include columns that are not in the select list. Both the *pub_id* and *type* columns are used to group the results for the vector aggregate. However, the final results do not include the type within each publisher. In this case, you may only want to know how many distinct title types are sold for each publisher.

```
7. select pub_id, count(pub_id)
   from publishers
```

```
pub_id
---------- ---------
0736               3
0877               3
1389               3

(3 rows affected)
```

This example combines two Transact-SQL extension effects. First, it omits the **group by** clause while including an aggregate in the select list. Second, it includes an extended column. By omitting the **group by** clause:

- The table becomes a single group. The scalar aggregate counts three qualified rows.
- *pub_id* becomes a Transact-SQL extended column because it does not appear in a **group by** clause. No **having** clause is present, so all rows in the group are qualified to be displayed.

```
8. select pub_id, count(pub_id)
   from publishers
   where pub_id < "1000"
```

```
pub_id
-------------- -----------
0736                     2
0877                     2
1389                     2

(3 rows affected)
```

The **where** clause excludes publishers with a *pub_id* of 1000 or more from the single group, so the scalar aggregate counts two qualified rows. The extended column *pub_id* displays all qualified rows from the *publishers* table.

```
9. select pub_id, count(pub_id)
   from publishers
   having pub_id < "1000"
```

```
pub_id
-------------- ---------
0736                   3
0877                   3
(2 rows affected)
```

This example illustrates an effect of a **having** clause used without a **group by** clause.

- The table is considered a single group. No **where** clause excludes rows, so all the rows in the group (table) are qualified to be counted.
- The rows in this single-group table are tested by the **having** clause.
- These combined effects display the two qualified rows.

```
10.select type, avg(price)
   from titles
   group by type
   having sum(total_sales) > 10000

   type
   ------------  ----------
   business           13.73
   mod_cook           11.49
   popular_comp       21.48
   trad_cook          15.96

   (4 rows affected)
```

This example uses the extension to **having** that allows columns or expressions not in the select list and not in the **group by** clause. It determines the average price for each title type, but it excludes those types that do not have more than $10,000 in total sales, even though the **sum** aggregate does not appear in the results.

group by and *having* and Sort Orders

- If your server has a case-insensitive sort order, **group by** ignores the case of the grouping columns. For example, given this data on a case-insensitive server:

```
select lname, amount
from groupdemo

lname      amount
---------- ------------------
Smith                  10.00
smith                   5.00
SMITH                   7.00
Levi                    9.00
Lévi                   20.00
```

grouping by *lname* produces these results:

```
select lname, sum(amount)
from groupdemo
group by lname
```

```
lname
---------- -----------------
Levi                    9.00
Lévi                   20.00
Smith                  22.00
```

The same query on a case- and accent-insensitive server produces these results:

```
lname
---------- -----------------
Levi                   29.00
Smith                  22.00
```

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of columns within the **select** list that are not in the **group by** list and have no aggregate functions is a Transact-SQL extension. The use of the **all** keyword is a Transact-SQL extension.

See Also

Commands	**compute Clause**, **declare**, **select**, **where Clause**
Functions	Aggregate Functions

if...else

Function

Imposes conditions on the execution of a SQL statement.

Syntax

```
if logical_expression [plan "abstract plan"]
   statements
[else
   [if logical_expression] [plan "abstract plan"]
   statement]
```

Keywords and Options

logical_expression – is an expression (a column name, a constant, any combination of column names and constants connected by arithmetic or bitwise operators, or a subquery) that returns TRUE, FALSE, or NULL. If the expression contains a select statement, the select statement must be enclosed in parentheses.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for optimizable SQL statements, that is, **select** queries that access tables.

statements – is either a single SQL statement or a block of statements delimited by **begin** and **end**.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for optimizable expressions in **if** clauses, that is, queries that access tables. For more information, see Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide*.

Examples

```
1. if 3 > 2
    print "yes"

2. if exists (select postalcode from authors
    where postalcode = "94705")
    print "Berkeley author"
```

```
3. if (select max(id) from sysobjects) < 100
      print "No user-created objects in this
   database" else
    begin
      print "These are the user-created objects"
      select name, type, id
      from sysobjects
      where id > 100
    end
```

The **if...else** condition tests for the presence of user-created objects (all of which have ID numbers greater than 100) in a database. Where user tables exist, the **else** clause prints a message and selects their names, types, and ID numbers.

```
4. if (select total_sales
       from titles
       where title_id = "PC9999") > 100
   select "true"
   else
   select "false"
```

Since the value for total sales for PC9999 in the *titles* table is NULL, this query returns FALSE. The **else** portion of the query is performed when the **if** portion returns FALSE or NULL. For more information on truth values and logical expressions, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Comments

- The statement following an **if** keyword and its condition is executed if the condition is satisfied (when the logical expression returns TRUE). The optional **else** keyword introduces an alternate SQL statement that executes when the **if** condition is not satisfied (when the logical expression returns FALSE).
- The **if** or **else** condition affects the performance of only a single SQL statement, unless statements are grouped into a block between the keywords **begin** and **end** (see example 3).

  The statement clause could be an **execute** stored procedure command or any other legal SQL statement or statement block.
- If a **select** statement is used as part of the boolean expression, it must return a single value.
- **if...else** constructs can be used either in a stored procedure (where they are often used to test for the existence of some parameter) or in *ad hoc* queries (see examples 1 and 2).

- **if** tests can be nested either within another **if** or following an **else**. The maximum number of **if** tests you can nest varies with the complexity of any **select** statements (or other language constructs) that you include with each **if...else** construct.

➤ ***Note***

When a **create table** or **create view** command occurs within an **if...else** block, Adaptive Server creates the schema for the table or view before determining whether the condition is true. This may lead to errors if the table or view already exists.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

if...else permission defaults to all users. No permission is required to use it.

See Also

Commands	**begin...end**, **create procedure**

insert

Function

Adds new rows to a table or view.

Syntax

```
insert [into]
   [database.[owner.]]{table_name|view_name}
   [(column_list)]
   {values (expression [, expression]...)
       |select_statement [plan "abstract plan"] }
```

Keywords and Options

into – is optional.

table_name | *view_name* – is the name of the table or view from which you want to remove rows. Specify the database name if the table or view is in another database, and specify the owner's name if more than one table or view of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

column_list – is a list of one or more columns to which data is to be added. Enclose the list in parentheses. The columns can be listed in any order, but the incoming data (whether in a **values** clause or a **select** clause) must be in the same order. If a column has the IDENTITY property, you can substitute the **syb_identity** keyword for the actual column name.

The column list is necessary when some, but not all, of the columns in the table are to receive data. If no column list is given, Adaptive Server assumes that the **insert** affects all columns in the receiving table (in **create table** order).

See "The Column List" for more information.

values – is a keyword that introduces a list of expressions.

expression – specifies constant expressions, variables, parameters or null values for the indicated columns. Enclose character and datetime constants in single or double quotes.

You cannot use a subquery as an *expression*.

The values list must be enclosed in parentheses and must match the explicit or implicit column list. See "Datatypes" for more information about data entry rules.

select_statement – is a standard **select** statement used to retrieve the values to be inserted.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for **insert...select** statements. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples

```
1. insert titles
   values("BU2222", "Faster!", "business", "1389",
       null, null, null, "ok", "06/17/87", 0)

2. insert titles
   (title_id, title, type, pub_id, notes, pubdate,
       contract)
   values ('BU1237', 'Get Going!', 'business',
        '1389', 'great', '06/18/86', 1)

3. insert newauthors
     select *
     from authors
     where city = "San Francisco"

4. insert test
     select *
     from test
     where city = "San Francisco"
```

Comments

- Use **insert** only to add new rows. Use **update** to modify column values in a row you have already inserted.

The Column List

- The column list determines the order in which values are entered. For example, suppose that you have a table called *newpublishers* that is identical in structure and content to the *publishers* table in *pubs2*. In the example below, the columns in the column list of the *newpublishers* table match the columns of the select list in the *publishers* table.

```
insert newpublishers (pub_id, pub_name)
select pub_id, pub_name
    from publishers
    where pub_name="New Age Data"
```

The *pub_id* and *pub_name* for "New Age Data" are stored in the *pub_id* and *pub_name* columns of *newpublishers*.

In the next example, the order of the columns in the column list of the *newpublishers* table does not match the order of the columns of the select list of the *publishers* table.

```
insert newpublishers (pub_id, pub_name)
    select pub_name, pub_id
    from publishers
    where pub_name="New Age Data"
```

The result is that the *pub_id* for "New Age Data" is stored in the *pub_name* column of the *newpublishers* table, and the *pub_name* for "New Age Data" is stored in the *pub_id* column of the *newpublishers* table.

- You can omit items from the column and values lists as long as the omitted columns allow null values (see example 2).

Validating Column Values

- **insert** interacts with the **ignore_dup_key**, **ignore_dup_row**, and **allow_dup_row** options, which are set with the **create index** command. (See **create index** for more information.)
- A rule or **check** constraint can restrict the domain of legal values that can be entered into a column. Rules are created with the **create rule** command and bound with the system procedure **sp_bindrule**. **check** constraints are declared with the **create table** statement.
- A default can supply a value if you do not explicitly enter one. Defaults are created with the **create default** command and bound with the system procedure **sp_bindefault**, or they are declared with the **create table** statement.
- If an **insert** statement violates domain or integrity rules (see **create rule** and **create trigger**), or if it is the wrong datatype (see **create table** and "System and User-Defined Datatypes"), the statement fails, and Adaptive Server displays an error message.

Treatment of Blanks

- Inserting an empty string ("") into a variable character type or *text* column inserts a single space. *char* columns are padded to the defined length.
- All trailing spaces are removed from data that is inserted into *varchar* columns, except in the case of a string that contains only spaces. Strings that contain only spaces are truncated to a single space. Strings that are longer than the specified length of a *char*, *nchar*, *varchar*, or *nvarchar* column are silently truncated unless the **string_rtruncation** option is set to **on**.

Inserting into *text* and *image* Columns

- An **insert** of a NULL into a *text* or an *image* column does not create a valid text pointer, nor does it preallocate 2K per value as would otherwise occur. Use **update** to get a valid text pointer for that column.

Insert Triggers

- You can define a trigger that takes a specified action when an **insert** command is issued on a specified table.

Using *insert* When Component Integration Services Is Enabled

- You can send an **insert** as a language event or as a parameterized dynamic statement to remote servers.

Inserting Rows Selected from Another Table

- You can select rows from a table and insert them into the same table in a single statement (see example 4).
- To insert data with **select** from a table that has null values in some fields into a table that does not allow null values, you must provide a substitute value for any NULL entries in the original table. For example, to insert data into an *advances* table that does not allow null values, substitute 0 for the NULL fields:

  ```
  insert advances
  select pub_id, isnull(advance, 0) from titles
  ```

  Without the **isnull** function, this command would insert all the rows with non-null values into the *advances* table, which would produce error messages for all the rows where the *advance* column in the *titles* table contained NULL.

If you cannot make this kind of substitution for your data, you cannot insert data containing null values into the columns that have a NOT NULL specification.

Two tables can be identically structured, and yet be different as to whether null values are permitted in some fields. You can use **sp_help** to see the null types of the columns in your table.

Transactions and *insert*

- When you set chained transaction mode, Adaptive Server implicitly begins a transaction with the **insert** statement if no transaction is currently active. To complete any inserts, you must commit the transaction, or roll back the changes. For example:

```
insert stores (stor_id, stor_name, city, state)
    values ('9999', 'Books-R-Us', 'Fremont', 'AZ')
if exists (select t1.city, t2.city
    from stores t1, stores t2
    where t1.city = t2.city
    and t1.state = t2.state
    and t1.stor_id < t2.stor_id)
        rollback transaction
else
        commit transaction
```

  In chained transaction mode, this batch begins a transaction and inserts a new row into the *stores* table. If it inserts a row containing the same city and state information as another store in the table, it rolls back the changes to *stores* and ends the transaction. Otherwise, it commits the insertions and ends the transaction. For more information about chained transaction mode, see the *Transact-SQL User's Guide*.

Inserting Values into IDENTITY Columns

- When inserting a row into a table, do not include the name of the IDENTITY column in the column list or its value in the values list. If the table consists of only one column, an IDENTITY column, omit the column list and leave the values list empty as follows:

```
insert id_table values()
```

- The first time you insert a row into a table, Adaptive Server assigns the IDENTITY column a value of 1. Each new row gets a column value that is one higher than the last. This value takes precedence over any defaults declared for the column in the **create table** or **alter table** statement or defaults bound to the column with the **sp_bindefault** system procedure.

Server failures can create gaps in IDENTITY column values. The maximum size of the gap depends on the setting of the **identity burning set factor** configuration parameter. Gaps can also result from manual insertion of data into the IDENTITY column, deletion of rows, and transaction rollbacks.

- Only the table owner, Database Owner, or System Administrator can explicitly insert a value into an IDENTITY column after setting **identity_insert** *table_name* **on** for the column's base table. A user can set **identity_insert** *table_name* **on** for one table at a time in a database. When **identity_insert** is **on**, each **insert** statement must include a column list and must specify an explicit value for the IDENTITY column.

  Inserting a value into the IDENTITY column allows you to specify a seed value for the column or to restore a row that was deleted in error. Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the value; you can insert any positive integer.

  To insert an explicit value into an IDENTITY column, the table owner, Database Owner, or System Administrator must set **identity_insert** *table_name* **on** for the column's base table, not for the view through which it is being inserted.

- The maximum value that can be inserted into an IDENTITY column is 10 $^{\text{PRECISION}}$ - 1. Once an IDENTITY column reaches this value, any additional **insert** statements return an error that aborts the current transaction.

  When this happens, use the **create table** statement to create a new table that is identical to the old one, but that has a larger precision for the IDENTITY column. Once you have created the new table, use either the **insert** statement or the **bcp** utility to copy the data from the old table to the new one.

- Use the *@@identity* global variable to retrieve the last value that you inserted into an IDENTITY column. If the last **insert** or **select into** statement affected a table with no IDENTITY column, *@@identity* returns the value 0.

- An IDENTITY column selected into a result table observes the following rules with regard to inheritance of the IDENTITY property:

  - If an IDENTITY column is selected more than once, it is defined as NOT NULL in the new table. It does not inherit the IDENTITY property.

- If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as NULL if any column in the expression allows nulls; otherwise, it is created as NOT NULL.
- If the select statement contains a **group by** clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created NULL; others are created NOT NULL.
- An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a NULL column, the new column is defined as NULL; otherwise, it is defined as NOT NULL.

Inserting Data Through Views

- If a view is created **with check option**, each row that is inserted through the view must meet the selection criteria of the view.

  For example, the *stores_cal* view includes all rows of the *stores* table for which *state* has a value of "CA":

  ```
  create view stores_cal
  as select * from stores
  where state = "CA"
  with check option
  ```

  The **with check option** clause checks each **insert** statement against the view's selection criteria. Rows for which *state* has a value other than "CA" are rejected.

- If a view is created **with check option**, all views derived from the **base** view must satisfy the view's selection criteria. Each new row inserted through a derived view must be visible through the base view.

  Consider the view *stores_cal30*, which is derived from *stores_cal*. The new view includes information about stores in California with payment terms of "Net 30":

  ```
  create view stores_cal30
  as select * from stores_cal
  where payterms = "Net 30"
  ```

  Because *stores_cal* was created **with check option**, all rows inserted or updated through *stores_cal30* must be visible through *stores_cal*. Any row with a *state* value other than "CA" is rejected.

Notice that *stores_cal30* does not have a **with check option** clause of its own. This means that it is possible to insert or update a row with a *payterms* value other than "Net 30" through *stores_cal30*. The following **update** statement would be successful, even though the row would no longer be visible through *stores_cal30*:

```
update stores_cal30
set payterms = "Net 60"
where stor_id = "7067"
```

- **insert** statements are not allowed on join views created **with check option.**
- If you insert or update a row through a join view, all affected columns must belong to the same base table.

Partitioning Tables for Improved Insert Performance

- An unpartitioned table with no clustered index consists of a single doubly linked chain of database pages, so each insertion into the table uses the last page of the chain. Adaptive Server holds an exclusive lock on the last page while it inserts the rows, blocking other concurrent transactions from inserting data into the table.

  Partitioning a table with the **partition** clause of the **alter table** command creates additional page chains. Each chain has its own last page, which can be used for concurrent insert operations. This improves insert performance by reducing page contention. If the table is spread over multiple physical devices, partitioning also improves insert performance by reducing I/O contention while the server flushes data from cache to disk. For more information about partitioning tables for insert performance, see the *Performance and Tuning Guide*.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The following are Transact-SQL extensions: • A **union** operator in the select portion of an **insert** statement • Qualification of a table or column name by a database name • Insertion through a view that contains a join **Note:** Insertion through a view that contains a join is not detected by the FIPS flagger.)

Permissions

insert permission defaults to the table or view owner, who can transfer it to other users.

insert permission for a table's IDENTITY column is limited to the table owner, Database Owner, and System Administrator.

See Also

Commands	**alter table, create default, create index, create rule, create table, create trigger, dbcc, delete, select, update**
Datatypes	"System and User-Defined Datatypes"
System procedures	**sp_bindefault, sp_bindrule, sp_help, sp_helpartition, sp_unbindefault, sp_unbindrule**
Utility programs	**bcp**

kill

Function

Kills a process.

Syntax

kill *spid*

Keywords and Options

spid – is the identification number of the process you want to kill. *spid* must be a constant; it cannot be passed as a parameter to a stored procedure or used as a local variable. Use **sp_who** to see a list of processes and other information.

Examples

1. kill 1378

Comments

- To get a report on the current processes, execute the system procedure **sp_who**. Following is a typical report:

```
fid spid  status    loginame origname hostname blk  dbname
      cmd
--- ----- --------   -------- -------- -------- --- ------
      --------------
  0    1  recv sleep  bird     bird     jazzy    0   master
      AWAITING COMMAND
  0    2  sleeping    NULL     NULL              0   master
      NETWORK HANDLER
  0    3  sleeping    NULL     NULL              0   master
      MIRROR HANDLER
  0    4  sleeping    NULL     NULL              0   master
      AUDIT PROCESS
  0    5  sleeping    NULL     NULL              0   master
      CHECKPOINT SLEEP
```

```
0    6  recv sleep  rose     rose     petal    0   master
     AWAITING COMMAND
0    7  running     robert   sa       helos    0   master
     SELECT
0    8  send sleep  daisy    daisy    chain    0   pubs2
     SELECT
0    9  alarm sleep lily     lily     pond     0   master
     WAITFOR
0   10  lock sleep  viola    viola    cello    7   pubs2
     SELECT
```

The *spid* column contains the process identification numbers used in the Transact-SQL **kill** command. The *blk* column contains the process ID of a blocking process, if there is one. A blocking process (which may have an exclusive lock) is one that is holding resources that are needed by another process. In this example, process 10 (a **select** on a table) is blocked by process 7 (a **begin transaction** followed by an **insert** on the same table).

- The *status* column reports the state of the command. The following table shows the status values and the effects of **sp_who**:

Table 6-26: Status values reported by sp_who

Status		Effect of *kill* Command
`recv sleep`	Waiting on a network read	Immediate.
`send sleep`	Waiting on a network send	Immediate.
`alarm sleep`	Waiting on an alarm, such as **waitfor delay "10:00"**	Immediate.
`lock sleep`	Waiting on a lock acquisition	Immediate.
`sleeping`	Waiting on disk I/O or some other resource. Probably indicates a process that is running, but doing extensive disk I/O	Killed when it "wakes up", usually immediate. A few sleeping processes do not wake up, and require a Adaptive Server reboot to clear.
`runnable`	In the queue of runnable processes	Immediate.
`running`	Actively running on one of the server engines	Immediate.
`infected`	Adaptive Server has detected a serious error condition; extremely rare	**kill** command not recommended. Adaptive Server restart probably required to clear process.

Table 6-26: Status values reported by sp_who (continued)

Status		Effect of *kill* Command
`background`	A process, such as a threshold procedure, run by Adaptive Server rather than by a user process	Immediate; use **kill** with extreme care. Recommend a careful check of *sysprocesses* before killing a background process.
`log suspend`	Processes suspended by reaching the last-chance threshold on the log	Immediate.

- To get a report on the current locks and the *spid*s of the processes holding them, use **sp_lock.**

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

kill permission defaults to System Administrators and is not transferable.

See Also

Commands	**shutdown**
System procedures	**sp_lock, sp_who**

load database

Function

Loads a backup copy of a user database, including its transaction log, that was created with **dump database**.

Syntax

```
load database database_name
   from stripe_device [at backup_server_name ]
       [density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name]
   [stripe on stripe_device [at backup_server_name ]
            [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [[stripe on stripe_device [at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       listonly [= full],
       headeronly,
       notify = {client | operator_console}
       }]]
```

Keywords and Options

database_name – is the name of the database that will receive the backup copy. It can be either a database created with the **for load** option, or an existing database. Loading dumped data to an existing database overwrites all existing data. The receiving database must be at least as large as the dumped database. The database name can be specified as a literal, a local variable, or a stored procedure parameter.

from *stripe_device* – is the device from which data is being loaded. See "Specifying Dump Devices" for information about what form to use when specifying a dump device. See the Adaptive Server installation and configuration guide for a list of supported dump devices.

at *backup_server_name* – is the name of a remote Backup Server running on the machine to which the dump device is attached. For platforms that use interfaces files, the *backup_server_name* must appear in the interfaces file.

density = *density_value* – is ignored.

blocksize = *number_bytes* – overrides the default block size for a dump device. **Do not specify a block size on OpenVMS systems.** If you specify a block size on UNIX systems, it should be identical to that used to make the dump.

dumpvolume = *volume_name* – is the volume name field of the ANSI tape label. **load database** checks this label when the tape is opened and generates an error message if the wrong volume is loaded.

stripe on *stripe_device* – is an additional dump device. You can use up to 32 devices, including the device named in the **to** *stripe_device* clause. The Backup Server loads data from all devices concurrently, reducing the time and the number of volume changes required. See "Specifying Dump Devices" for information about how to specify a dump device.

dismount | **nodismount** – **on platforms that support logical dismount** (such as OpenVMS), determines whether tapes remain mounted. By default, all tapes used for a load are dismounted when the load completes. Use **nodismount** to keep tapes available for additional loads or dumps.

nounload | **unload** – determines whether tapes rewind after the load completes. By default, tapes do not rewind, allowing you to make additional loads from the same tape volume. Specify **unload** for the last dump file to be loaded from a multi-dump volume. This rewinds and unloads the tape when the load completes.

file = *file_name* – is the name of a particular database dump on the tape volume. If you did not record the dump file names at the time you made the dump, use **listonly** to display information about all dump files.

listonly [**= full**] – displays information about all dump files on a tape volume, but **does not load the database**. **listonly** identifies the database and device, the date and time the dump was made, and the date and time it can be overwritten. **listonly = full** provides additional details about the dump. Both reports are sorted by ANSI tape label.

After listing the files on a volume, the Backup Server sends a volume change request. The operator can either mount another tape volume or terminate the list operation for all dump devices.

Due to current implementation, the **listonly** option overrides the **headeronly** option.

◆ ***WARNING!***

Do not use load database with listonly on 1/4-inch cartridge tape.

headeronly – displays header information for a single dump file, but **does not load the database**. **headeronly** displays information about the first file on the tape unless you use the **file =** *file_name* option to specify another file name. The dump header indicates:

- Type of dump (database or transaction log)
- Database ID
- File name
- Date the dump was made
- Character set
- Sort order
- Page count
- Next object ID

notify = {**client** | **operator_console**} – overrides the default message destination.

- On operating systems (such as OpenVMS) that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use **client** to route other Backup Server messages to the terminal session that initiated the **dump database**.
- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that

initiated the **dump database**. Use **operator_console** to route messages to the terminal on which the Backup Server is running.

Examples

1. **For UNIX:**

```
load database pubs2
  from "/dev/nrmt0"
```

For OpenVMS:

```
load database pubs2
  from "MTA0:"
```

Reloads the database *pubs2* from a tape device.

2. **For UNIX:**

```
load database pubs2
    from "/dev/nrmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
      stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
```

For OpenVMS:

```
load database pubs2
    from "MTA0:" at REMOTE_BKP_SERVER
    stripe on "MTA1:" at REMOTE_BKP_SERVER
  stripe on "MTA2:" at REMOTE_BKP_SERVER
```

Loads the *pubs2* database, using the Backup Server REMOTE_BKP_SERVER. This command names three devices.

Comments

- The **listonly** and **headeronly** options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
- Table 6-27 describes the commands and system procedures used to restore databases from backups:

Table 6-27: Commands used to restore databases from dumps

Use This Command	To Do This
create database for load	Create a database for the purpose of loading a dump.
load database	Restore a database from a dump.
load transaction	Apply recent transactions to a restored database.

Table 6-27: Commands used to restore databases from dumps (continued)

Use This Command	To Do This
online database	Make a database available for public use after a normal load sequence or after upgrading the database to the current version of Adaptive Server.
load { database | transaction } with {headeronly | listonly}	Identify the dump files on a tape.
sp_volchanged	Respond to Backup Server's volume change messages.

load database Restrictions

- You cannot load a dump that was made on a different platform.
- You cannot load a dump that was generated on a pre-release 10.0 server.
- If a database has cross-database referential integrity constraints, the *sysreferences* system table stores the **name**—not the ID number—of the external database. Adaptive Server cannot guarantee referential integrity if you use **load database** to change the database name or to load it onto a different server.
- Each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of these databases can cause database corruption. Before dumping a database in order to load it with a different name or move it to another Adaptive Server, use alter table to drop all external referential integrity constraints.

- **load database** clears the suspect page entries pertaining to the loaded database from *master..sysattributes*.
- **load database** overwrites any existing data in the database.
- After a database dump is loaded, two processes may require additional time before the database can be used:
  - If the database was created with the **for load** option to **create database**, all unused pages in the database must be zeroed after

the load completes. The time required depends on the number of unused pages.

- All transactions in the transaction log included in the database dump must be rolled back or rolled forward. The time required depends on the number and type of transactions in the log.

- The receiving database must be as large as or larger than the database to be loaded. If the receiving database is too small, Adaptive Server displays an error message that gives the required size.
- You cannot load from the null device (on UNIX, */dev/null*; on OpenVMS, any device name beginning with "NL").
- You cannot use the **load database** command in a user-defined transaction.

Locking Out Users During Loads

- While you are loading a database, it cannot be in use. The **load database** command sets the status of the database to "offline." No one can use the database while its status is "offline". The "offline" status prevents users from accessing and changing the database during a load sequence.
- A database loaded by **load database** remains inaccessible until the **online database** command is issued.

Upgrading Database and Transaction Log Dumps

- To restore and upgrade a user database dump from a release 10.0 or later server to the current release of Adaptive Server:

  1. Load the most recent database dump.
  2. Load, **in order,** all transaction log dumps made since the last database dump.

     Adaptive Server checks the timestamp on each dump to make sure that it is being loaded to the correct database and in the correct sequence.
  3. Issue the **online database** command to do the upgrade and make the database available for public use.
  4. Dump the newly upgraded database immediately after upgrade, to create a dump consistent with the current release of Adaptive Server.

Specifying Dump Devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- You can specify a local device as:
  - A logical device name from the *sysdevices* system table
  - An absolute path name
  - A relative path name

  The Backup Server resolves relative path names using Adaptive Server's current working directory.
- When loading across the network, specify the absolute path name of the dump device. The path name must be valid on the machine on which the Backup Server is running. If the name includes characters other than letters, numbers, or the underscore (_), enclose the entire name in quotes.
- Ownership and permissions problems on the dump device may interfere with use of load commands.
- You can run more than one load (or dump) at the same time, as long as each load uses a different physical device.

Backup Servers

- You must have a Backup Server running on the same machine as Adaptive Server. (On OpenVMS systems, the Backup Server can be running in the same cluster as the Adaptive Server, as long as all database devices are visible to both servers.) The Backup Server must be listed in the *master..sysservers* table. This entry is created during installation or upgrade and should not be deleted.
- If your backup devices are located on another machine, so that you load across a network, you must also have a Backup Server installed on the remote machine.

Volume Names

- Dump volumes are labeled according to the ANSI tape labeling standard. The label includes the logical volume number and the position of the device within the stripe set.
- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

➤ ***Note***

When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing Dump Volumes

- If the Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies the Backup Server by executing the **sp_volchanged** system procedure on any Adaptive Server that can communicate with the Backup Server.
- On OpenVMS systems, the operating system requests a volume change when the specified drive is offline. After mounting another volume, the operator uses the REPLY command to reply to volume change messages.

Restoring the System Databases

- See the *System Administration Guide* for step-by-step instructions for restoring the system databases from dumps.

Disk Mirroring

- At the beginning of a load, Adaptive Server passes Backup Server the primary device name of each logical database and log device. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before Backup Server completes its data transfer, Adaptive Server aborts the load.
- If you attempt to unmirror any named device while a **load database** is in progress, Adaptive Server displays a message. The user executing the **disk unmirror** command can abort the load or defer the **disk unmirror** until after the load completes.
- Backup Server loads the data onto the primary device, then **load database** copies it to the secondary device. **load database** takes longer to complete if any database device is mirrored.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Administrator, Database Owner, or user with the Operator role can execute **load database**.

See Also

Commands	dbcc, dump database, dump transaction, load transaction, online database
System procedures	sp_helpdevice, sp_volchanged, sp_helpdb

load transaction

Function

Loads a backup copy of the transaction log that was created with the **dump transaction** command.

Syntax

```
load tran[saction] database_name
   from stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [[stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       listonly [= full],
       headeronly,
       notify = {client | operator_console}
       until_time = datetime}]]
```

Keywords and Options

database_name – is the name of the database that will receive data from a dumped backup copy of the transaction log. The log segment of the receiving database must be at least as large as the log segment of the dumped database. The database name can be specified as a literal, a local variable, or a parameter of a stored procedure.

from *stripe_device* – is the name of the dump device from which you are loading the transaction log. See "Specifying Dump Devices" for information about what form to use when specifying a dump device. See the Adaptive Server installation and configuration guide for a list of supported dump devices.

at *backup_server_name* – is the name of a remote Backup Server running on the machine to which the dump device is attached. For platforms that use interfaces files, the *backup_server_name* must appear in the interfaces file.

density = *density_value* – overrides the default density for a tape device. **This option is ignored.**

blocksize = *number_bytes* – overrides the default block size for a dump device. Do not specify a block size on OpenVMS systems. If you specify a block size on UNIX systems, it should be identical to that used to make the dump.

dumpvolume = *volume_name* – is the volume name field of the ANSI tape label. **load transaction** checks this label when the tape is opened and generates an error message if the wrong volume is loaded.

stripe on *stripe_device* – is an additional dump device. You can use up to 32 devices, including the device named in the **to** *stripe_device* clause. The Backup Server loads data from all devices concurrently, reducing the time and the number of volume changes required. See "Specifying Dump Devices" for information about how to specify a dump device.

dismount | **nodismount** – **on platforms that support logical dismount** (such as OpenVMS), determines whether tapes remain mounted. By default, all tapes used for a load are dismounted when the load completes. Use **nodismount** to keep tapes available for additional loads or dumps.

nounload | **unload** – determines whether tapes rewind after the load completes. By default, tapes do not rewind, allowing you to make additional loads from the same tape volume. Specify **unload** for the last dump file to be loaded from a multidump volume. This rewinds and unloads the tape when the load completes.

file = *file_name* – is the name of a particular database dump on the tape volume. If you did not record the dump file names at the time you made the dump, use **listonly** to display information about all the dump files.

listonly [**= full**] – displays information about all the dump files on a tape volume, but **does not load the transaction log**. **listonly** identifies the database and device, the date and time the dump was made, and the date and time it can be overwritten. **listonly = full** provides additional details about the dump. Both reports are sorted by ANSI tape label.

After listing the files on a volume, the Backup Server sends a volume change request. The operator can either mount another tape volume or terminate the list operation for all dump devices.

Due to current implementation, the **listonly** option overrides the **headeronly** option.

◆ ***WARNING!***

Do not use load transaction with listonly on 1/4-inch cartridge tape.

headeronly – displays header information for a single dump file, but **does not load the database**. **headeronly** displays information about the first file on the tape unless you use the **file =** *file_name* option to specify another file name. The dump header indicates:

- Type of dump (database or transaction log)
- Database ID
- File name
- Date the dump was made
- Character set
- Sort order
- Page count
- Next object ID
- Checkpoint location in the log
- Location of the oldest **begin transaction** record
- Old and new sequence dates

notify = {**client** | **operator_console**} – overrides the default message destination.

- On operating systems (such as OpenVMS) that offer an operator terminal feature, volume change messages are always sent to the operator terminal on the machine on which the Backup Server is running. Use **client** to route other Backup

Server messages to the terminal session that initiated the **dump database**.

- On operating systems (such as UNIX) that do not offer an operator terminal feature, messages are sent to the client that initiated the **dump database**. Use **operator_console** to route messages to the terminal on which the Backup Server is running.

until_time – loads the transaction log up to a specified time in the transaction log. Only transactions committed before the specified time are saved to the database.

Examples

1. **For UNIX:**
   ```
   load transaction pubs2
     from "/dev/nrmt0"
   ```
   For OpenVMS:
   ```
   load transaction pubs2
     from "MTA0:"
   ```
   Loads the transaction log for the database *pubs2* tape.

2. **For UNIX:**
   ```
   load transaction pubs2
         from "/dev/nrmt4" at REMOTE_BKP_SERVER
       stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
       stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
   ```
   For OpenVMS:
   ```
   load transaction pubs2
     from "MTA0:" at REMOTE_BKP_SERVER
     stripe on "MTA1:" at REMOTE_BKP_SERVER
     stripe on "MTA2:" at REMOTE_BKP_SERVER
   ```
   Loads the transaction log for the *pubs2* database, using the Backup Server REMOTE_BKP_SERVER.

3. ```
   load transaction pubs2
   from "/dev/ntmt0"
   with until_time = "mar 20, 1997 10:51:43:866am"
   ```
   Loads the transaction log for *pubs2*, up to March 20, 1997, at 10:51:43:866 a.m.

Comments

- The **listonly** and **headeronly** options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
   ```

- Table 6-28 describes the commands and system procedures used to restore databases from backups:

Table 6-28: Commands used to restore databases

Use This Command	To Do This
create database for load	Create a database for the purpose of loading a dump.
load database	Restore a database from a dump.
load transaction	Apply recent transactions to a restored database.
online database	Make a database available for public use after a normal load sequence or after upgrading the database to the current version of Adaptive Server.
load { database | transaction } with {headeronly | listonly}	Identify the dump files on a tape.
sp_volchanged	Respond to the Backup Server's volume change messages.

load transaction Restrictions

- You cannot load a dump that was made on a different platform.
- You cannot load a dump that was generated on a pre-release 10.0 server.
- The database and transaction logs must be at the same release level.
- Load transaction logs in chronological order.
- You cannot load from the null device (on UNIX, */dev/null*; on OpenVMS, any device name beginning with "NL").
- You cannot use **load transaction** after an **online database** command that does an upgrade. The following sequence is **incorrect** for upgrading a database: **load database**, **online database**, **load transaction.** The correct sequence for upgrading a database is **load database**, **load transaction**, **online database**.
- You can use **load transaction** after **online database** if there was no upgrade or version change.
- You cannot use the **load transaction** command in a user-defined transaction.

Restoring a Database

- To restore a database:
  - Load the most recent database dump
  - Load, **in order**, all transaction log dumps made since the last database dump
  - Issue the **online database** command to make the database available for public use.
- Each time you add or remove a cross-database constraint, or drop a table that contains a cross-database constraint, dump **both** of the affected databases.

◆ ***WARNING!***

Loading earlier dumps of these databases can cause database corruption.

- For more information on backup and recovery of Adaptive Server databases, see the *System Administration Guide*.

Recovering a Database to a Specified Time

- You can use the **until_time** option for most databases that can be loaded or dumped. It does not apply to databases such as *master*, in which the data and logs are on the same device. Also, you cannot use it on any database that has had a truncated log since the last **dump database**, such as *tempdb*.
- The **until_time** option is useful for the following reasons:
  - It enables you to have a database consistent to a particular time. For example, in an environment with a Decision Support System (DSS) database and an Online Transaction Processing (OLTP) database, the System Administrator can roll the DSS database to an earlier specified time to compare data between the earlier version and the current version.
  - If a user inadvertently destroys data, such as dropping an important table, you can use the **until_time** option to back out the errant command by rolling forward the database to a point just before the data was destroyed.
- To effectively use the **until_time** option after data has been destroyed, you must know the exact time the error took place. You can find out by executing a **select getdate()** command

immediately after the error. For a more precise time using milliseconds, use the **convert** function, for example:

```
select convert(char(26), getdate(), 109)

--------------------------
Feb 26 1997 12:45:59:650PM
```

- After you load a transaction log using **until_time**, Adaptive Server restarts the database's log sequence. This means that until you dump the database again, you cannot load subsequent transaction logs after the **load transaction** using **until_time**. You will need to dump the database before you can dump another transaction log.
- Only transactions that committed before the specified time are saved to the database. However, in some cases, transactions committed shortly after the **until_time** specification are applied to the database data. This may occur when several transactions are committing at the same time. The ordering of transactions may not be written to the transaction log in time-ordered sequence. In this case, the transactions that are out of time sequence will be reflected in the data that has been recovered. The time should be less than a second.
- For more information on recovering a database to a specified time, see the *System Administration Guide.*

Locking Users out During Loads

- While you are loading a database, it cannot be in use. The **load transaction** command, unlike **load database**, does not change the offline / online status of the database. **load transaction** leaves the status of the database the way it found it. The **load database** command sets the status of the database to "offline". No one can use the database while it is "offline". The "offline" status prevents users from accessing and changing the database during a load sequence.
- A database loaded by **load database** remains inaccessible until the **online database** command is issued.

Upgrading Database and Transaction Log Dumps

- To restore and upgrade a user database dump from a release 10.0 or later server to the current release of Adaptive Server:

  1. Load the most recent database dump.

2. Load, **in order**, all transaction logs generated after the last database dump.
3. Use **online database** to do the upgrade.
4. Dump the newly upgraded database immediately after upgrade, to create a dump that is consistent with the current release of Adaptive Server.

Specifying Dump Devices

- You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.
- When loading from a local device, you can specify the dump device as:
  - An absolute path name
  - A relative path name
  - A logical device name from the *sysdevices* system table

  Backup Server resolves relative path names, using Adaptive Server's current working directory.
- When loading across the network, you must specify the absolute path name of the dump device. (You cannot use a relative path name or a logical device name from the *sysdevices* system table.) The path name must be valid on the machine on which the Backup Server is running. If the name includes any characters other than letters, numbers or the underscore (_), you must enclose it in quotes.
- Ownership and permissions problems on the dump device may interfere with use of load commands. The **sp_addumpdevice** procedure adds the device to the system tables, but does not guarantee that you can load from that device or create a file as a dump device.
- You can run more than one load (or dump) at the same time, as long as each one uses a different physical device.

Backup Servers

- You must have a Backup Server running on the same machine as your Adaptive Server. (On OpenVMS systems, Backup Server can be running in the same cluster as Adaptive Server, as long as all database devices are visible to both servers.) The Backup Server must be listed in the *master..sysservers* table. This entry is created during installation or upgrade and should not be deleted.

- If your backup devices are located on another machine so that you load across a network, you must also have a Backup Server installed on the remote machine.

Volume Names

- Dump volumes are labeled according to the ANSI tape-labeling standard. The label includes the logical volume number and the position of the device within the stripe set.
- During loads, Backup Server uses the tape label to verify that volumes are mounted in the correct order. This allows you to load from a smaller number of devices than you used at dump time.

➤ ***Note***

When dumping and loading across a network, you must specify the same number of stripe devices for each operation.

Changing Dump Volumes

- If Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies Backup Server by executing **sp_volchanged** on any Adaptive Server that can communicate with Backup Server.
- On OpenVMS systems, the operating system requests a volume change when the specified drive is offline. After mounting another volume, the operator uses the **REPLY** command to reply to volume change messages.

Restoring the System Databases

- See the *System Administration Guide* for step-by-step instructions for restoring the system databases from dumps.

Disk Mirroring

- At the beginning of a load, Adaptive Server passes the primary device name of each logical database device and each logical log device to the Backup Server. If the primary device has been unmirrored, Adaptive Server passes the name of the secondary device instead. If any named device fails before the Backup

Server completes its data transfer, Adaptive Server aborts the load.

- If you attempt to unmirror any of the named devices while a **load transaction** is in progress, Adaptive Server displays a message. The user executing the **disk unmirror** command can abort the load or defer the **disk unmirror** until after the load completes.
- Backup Server loads the data onto the primary device, then **load transaction** copies it to the secondary device. **load transaction** takes longer to complete if any database device is mirrored.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

load transaction permission defaults to the Database Owner and Operators. It is not transferable.

See Also

Commands	disk unmirror, dump database, dump transaction, load database, online database
System procedures	sp_dboption, sp_helpdb, sp_helpdevice, sp_volchanged

lock table

Function

Explicitly locks a table within a transaction.

Syntax

```
lock table table_name in {share | exclusive } mode
      [ wait [ numsecs ] | nowait ]
```

Keywords and Options

table_name – specifies the name of the table to be locked.

share | exclusive – specifies the type of lock, shared or exclusive, to be applied to the table.

wait *numsecs* – specifies the number of seconds to wait, if a lock cannot be acquired immediately. If *numsecs* is omitted, specifies that the **lock table** command should wait until lock is granted.

nowait – causes the command to fail if the lock cannot be acquired immediately.

Examples

1.
```
begin transaction
lock table titles in share mode
```

   Tries to acquire a shared table lock on the *titles* table. If a session-level wait has been set with the **set lock wait** command, the lock table command waits for that period of time; otherwise, the server-level wait period is used.

2.
```
begin transaction
lock table authors in exclusive mode wait 5
```

   Tries to acquire an exclusive table lock on the *authors* table. If the lock cannot be acquired within 5 seconds, the command returns an informational message. Subsequent commands within the transaction continue as they would have without the **lock table** command.

```
3. create procedure bigbatch
   as
   begin transaction
   lock table titles in share mode wait 5
   if @@error = 12207
   begin
         /*
         ** Allow SA to run without the table lock
         ** Other users get an error message
         */
          if (proc_role("sa_role") = 0)
          begin
          print "You cannot run this procedure at
             this time, please try again later"
          rollback transaction
          return 100
          end
   else
          begin
          print "Couldn't obtain table lock,
             proceeding with default locking."
          end
   end
   /* more SQL here */
   commit transaction
```

If a table lock is not acquired within 5 seconds, the procedure checks the user's role. If the procedure is executed by a user with **sa_role**, the procedure prints an advisory message and proceeds without a table lock. If the user does not have **sa_role**, the transaction is rolled back.

Comments

- You can use **lock table** only within a transaction. The table lock is held for the duration of the transaction.
- The behavior of **lock table** depends on the wait-time options that are specified in the command or that are active at the session level or server level.
- If the **wait** and **nowait** option are not specified, the **lock table** command uses either the session-level wait period or the server-level wait period. If a session-level wait has been set using the **set lock wait** command, it is used, otherwise, the server-level wait period is used.
- If the table lock cannot be obtained with the time limit (if any), the **lock table** command returns message 12207. The transaction is not

rolled back. Subsequent commands in the transaction proceed as they would have without the **lock table** command.

- You cannot use **lock table** on system tables or temporary tables.
- You can issue multiple **lock table** commands in the same transaction.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must have **select** access permission on the table to use **lock table in share mode**. You must have either **delete**, **insert**, or **update** access permission on the table to use **lock table in exclusive mode**.

See Also

Commands	set

nullif

Function

Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a **case** expression.

Syntax

nullif(*expression*, *expression*)

Keywords and Options

nullif – compares the values of the two expressions. If the first expression equals the second expression, **nullif** returns NULL. If the first expression does not equal the second expression, **nullif** returns the first expression.

expression – is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

```
1. select title,
       nullif(type, "UNDECIDED")
   from titles
```

Selects the *titles* and *type* from the *titles* table. If the book type is UNDECIDED, **nullif** returns a NULL value.

```
2. select title,
       case
           when type = "UNDECIDED" then NULL
           else type
       end
   from titles
```

This is an alternative way of writing example 1.

Comments

- **nullif** expression alternate for a **case** expression.
- **nullif** expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a **when...then** construct.

- **nullif** expressions can be used anywhere an expression can be used in SQL.
- At least one result of the **case** expression must return a non-null value. For example:

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

  results in the following error message:

```
All result expressions in a CASE expression must not be NULL.
```

- If your query produces a variety of datatypes, the datatype of a **case** expression result is determined by datatype hierarchy, as described in "Datatype of Mixed-Mode Expressions" in Chapter 1, "System and User-Defined Datatypes." If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, *char* and *int*), the query fails.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

nullif permission defaults to all users. No permission is required to use it.

See Also

Commands	**case**, **coalesce**, **select**, **if...else**, **where Clause**

online database

Function

Marks a database available for public use after a normal load sequence; if needed, upgrades a loaded database to the current release of Adaptive Server; brings a database online after loading a transaction log dumped with the **standby_access** option.

Syntax

```
online database database_name [for standby_access]
```

Parameters

database_name – specifies the name of the database to be brought online.

for standby_access – brings the database online on the assumption that the database contains no open transactions.

Examples

1. `online database pubs2`

   Makes the *pubs2* database available for public use after a load sequence completes.

2. `online database inventory_db for standby_access`

   Brings the database *inventory_db* online. Used after loading *inventory_db* with a transaction-log dump obtained through **dump tran...with standby_access**.

Comments

- The **online database** command brings a database online for general use after a normal database or transaction log load sequence.
- When a **load database** command is issued, the database's status is set to "offline." The offline status is set in the *sysdatabases* system table and remains set until the **online database** command completes.
- Do **not** issue the **online database** command until all transaction logs are loaded. The command sequence is:
  - **load database**
  - **load transaction** (there may be more than one **load transaction**)
  - **online database**

- If you execute **online database** against a currently online database, no processing occurs and no error messages are generated.
- **online database...for standby_access** can only be used with a transaction log that was dumped using **dump transaction...with standby_access**. If you use **online database...for standby_access** after loading a transaction log that was dumped without using **dump transaction...with standby access**, the **online database** command will generate an error message and fail.
- You can use **sp_helpdb** to find out whether a database is currently online, online for standby access, or offline.

Upgrading Databases

- **online database** initiates, if needed, the upgrade of a loaded database and transaction log dumps to make the database compatible with the current release of Adaptive Server. After the upgrade completes, the database is made available for public use. If errors occur during processing, the database remains offline.
- **online database** is required only after a database or transaction log load sequence. It is not required for new installations or upgrades. When Adaptive Server is upgraded to a new release, all databases associated with that server are automatically upgraded.
- **online database** only upgrades release 10.0 or later user databases.
- After you upgrade a database with **online database**, dump the newly upgraded database to create a dump that is consistent with the current release of Adaptive Server. You must dump the upgraded database before you can issue a **dump transaction** command.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Administrator, Database Owner, or user with the Operator role can execute **online database**.

See Also

Commands	**dump database, dump transaction, load database, load transaction**
System Procedures	**sp_helpdb**

open

Function

Opens a cursor for processing.

Syntax

open *cursor_name*

Parameters

cursor_name – is the name of the cursor to open.

Examples

1. open authors_crsr

Opens the cursor named *authors_crsr*.

Comments

- **open** opens a cursor. Cursors allow you to modify or delete rows on an individual basis. You must first open a cursor to use the **fetch**, **update**, and **delete** statements. For more information about cursors, see the *Transact-SQL User's Guide*.
- Adaptive Server returns an error message if the cursor is already open or if the cursor has not been created with the **declare cursor** statement.
- Opening the cursor causes Adaptive Server to evaluate the **select** statement that defines the cursor (specified in the **declare cursor** statement) and makes the cursor result set available for processing.
- When the cursor is first opened, it is positioned before the first row of the cursor result set.
- When you set the chained transaction mode, Adaptive Server implicitly begins a transaction with the **open** statement if no transaction is currently active.

Permissions

open permission defaults to all users.

Standards and Compliance

Standard	Compliance Level
SQL92	Entry level compliant

See Also

Commands	close, declare cursor, fetch

order by Clause

Function

Returns query results in the specified column(s) in sorted order.

Syntax

```
[Start of select statement]
[order by {[table_name.| view_name.]column_name
       | select_list_number | expression} [asc | desc]
   [,{[table_name.| view_name.] column_name
         select_list_number|expression} [asc
            |desc]]...]
[End of select statement]
```

Keywords and Options

order by – sorts the results by columns.

asc – sorts the results in ascending order. If you do not specify **asc** or **desc**, **asc** is assumed.

desc – sorts the results in descending order.

Examples

```
1. select title, type, price
   from titles
   where price > $19.99
   order by title
```

```
title
        type         price
------------------------------------------------------------
        ------------ -------------------------
But Is It User Friendly?
         popular_comp                   22.95
Computer Phobic and Non-Phobic Individuals: Behavior Variations
          psychology                     21.59
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
         trad_cook                      20.95
Secrets of Silicon Valley
         popular_comp                   20.00
```

Selects the titles whose price is greater than $19.99 and lists them with the titles in alphabetical order.

2.
```
select type, price, advance
from titles
order by type desc
compute avg(price), avg(advance) by type
```

   Lists the books from the *titles* table, in descending alphabetical order of the type, and calculates the average price and advance for each type.

3.
```
select title_id, advance/total_sales
from titles
order by advance/total_sales
```

```
 title_id
 -------- ------------------------
 MC3026                      NULL
 PC9999                      NULL
 MC2222                      0.00
 TC4203                      0.26
 PS3333                      0.49
 BU2075                      0.54
 MC3021                      0.67
 PC1035                      0.80
 PS2091                      1.11
 PS7777                      1.20
 BU1032                      1.22
 BU7832                      1.22
 BU1111                      1.29
 PC8888                      1.95
 TC7777                      1.95
 PS1372                     18.67
 TC3218                     18.67
 PS2106                     54.05
```

   Lists the title IDs from the *titles* table, with the advances divided by the total sales, ordered from the lowest calculated amount to the highest.

4.
```
select title as BookName, type as Type
from titles
order by Type
```

   Lists book titles and types in order by the type, renaming the columns in the output.

Comments

- **order by** returns query results in the specified column(s) in sorted order. **order by** is part of the **select** command.

- In Transact-SQL, you can use **order by** to sort items that do not appear in the select list. You can sort by a column heading, a column name, an expression, an alias name (if specified in the select list), or a number representing the position of the item in the select list (*select_list_number*).
- If you sort by *select_list_number*, the columns to which the *order by* clause refers must be included in the **select** list, and the **select** list cannot be * (asterisk).
- Use **order by** to display your query results in a meaningful order. Without an **order by** clause, you cannot control the order in which Adaptive Server returns results.

Restrictions

- The maximum number of columns allowed in an **order by** clause is 31.
- The sum of the maximum lengths of all the columns specified by the **order by** clause cannot exceed 2014 bytes.
- **order by** cannot be used on *text* or *image* datatype columns.
- Subqueries and view definitions cannot include an **order by** clause (or a **compute** clause or the keyword **into**). Conversely, you cannot use a subquery in an **order by** list.
- You cannot update the result set of a server- or language- type cursor if it contains an **order by** clause in its **select** statement. For more information about the restrictions applied to updatable cursors, see the *Transact-SQL User's Guide*.
- If you use **compute by**, you must also use an **order by** clause. The expressions listed after **compute by** must be identical to or a subset of those listed after **order by**, must be in the same left-to-right order, must start with the same expression, and must not skip any expressions. For example, if the **order by** clause is:

  ```
  order by a, b, c
  ```

  the **compute by** clause can be any (or all) of these:

  ```
  compute by a, b, c
  compute by a, b
  compute by a
  ```

  The keyword **compute** can be used without **by** to generate grand totals, grand counts, and so on. In this case, **order by** is optional.

Collating Sequences

- With **order by**, null values precede all others.
- The sort order (collating sequence) on your Adaptive Server determines how your data is sorted. The sort order choices are binary, dictionary, case-insensitive, case-insensitive with preference, and case- and accent-insensitive. Sort orders that are specific to specific national languages may also be provided.

Table 6-29: Effect of sort order choices

Adaptive Server Sort Order	Effects on *order by* Results
Binary order	Sorts all data according to the numeric byte-value of each character in the character set. Binary order sorts all uppercase letters before lowercase letters. Binary sort order is the only option for multibyte character sets.
Dictionary order	Sorts uppercase letters before their lowercase counterparts (case-sensitive). Dictionary order recognizes the various accented forms of a letter and sorts them after the unaccented form.
Dictionary order, case-insensitive	Sorts data in dictionary order but does not recognize case differences. Uppercase letters are equivalent to their lowercase counterparts and are sorted as described in “Sort Rules”.
Dictionary order, case-insensitive with preference	Sorts an uppercase letter in the preferred position, before its lowercase version. It does not recognize case difference when performing comparisons (for example, in where clauses).
Dictionary order, case- and accent-insensitive	Sorts data in dictionary order, but does not recognize case differences; treats accented forms of a letter as equivalent to the associated unaccented letter. It intermingles accented and unaccented letters in sorting results.

- The system procedure **sp_helpsort** reports the sort order installed on Adaptive Server.

Sort Rules

- When two rows have equivalent values in Adaptive Server's sort order, the following rules are used to order the rows:
  - The values in the columns named in the **order by** clause are compared.
  - If two rows have equivalent column values, the binary value of the entire rows is compared byte by byte. This comparison is performed on the row in the order in which the columns are stored internally, not the order of the columns as they are named in the query or in the original **create table** clause. (In brief, data is stored with all the fixed-length columns, in order, followed by all the variable length columns, in order.)
  - If rows are equal, row IDs are compared.

  Given this table:

```
create table sortdemo (lname varchar(20),
                       init char(1) not null)
```

  and this data:

```
lname      init
---------- ----
Smith      B
SMITH      C
smith      A
```

  you get these results when you order by *lname*:

```
lname      init
---------- ----
smith      A
Smith      B
SMITH      C
```

  Since the fixed-length *char* data (the *init* column) is stored first internally, the **order by** sorts these rows based on the binary values "Asmith", "BSmith" and "CSMITH".

  However, if the *init* is of type *varchar*, the *lname* column is stored first, and then the *init* column. The comparison takes place on the binary values "SMITHC", "SmithB", and "smithA", and the rows are returned in that order.

Descending Scans

- Use of the keyword **desc** in an **order by** clause allows the query optimizer to choose a strategy that eliminates the need for a

worktable and a sort step to return results in descending order. This optimization scans the page chain of the index in reverse order, following the previous page pointers on each index page.

In order to use this optimization, the columns in the **order by** clause must match the index order. They can be a subset of the keys, but must be a prefix subset, that is, they must include the first key(s). The descending scan optimization cannot be used if the columns named in the **order by** clause are a superset of the index keys.

If the query involves a join, all tables can be scanned in descending key order, as long as the requirements for a prefix subset of keys are met. Descending scan optimization can also be used for one or more tables in a join, while other tables are scanned in ascending order.

- If other user processes are scanning forward to perform updates or deletes, performing descending scans can cause deadlocks. Deadlocks may also be encountered during page splits and shrinks. You can use the system procedure **sp_sysmon** to track deadlocks on your server, or you can use the configuration parameter **print deadlock information** to send deadlock information to the error log.
- If your applications need to return results in descending order, but the descending scans optimization creates deadlock problems, some possible workarounds are:
  - Use transaction isolation level 0 scans for descending scans. For more information on the effect of isolation level 0 reads, see the *Performance and Tuning Guide*.
  - Disable descending scan optimization with the configuration parameter **allow backward scans** so that all queries that use **desc** will scan the table in ascending order and sort the result set into descending order. For more information, see the *System Administration Guide*.
  - Break problematical descending scans into two steps, selecting the required rows into a temporary table in ascending order in the first step, and selecting from the temporary table in descending order in the second step.
- If a backward scan uses a clustered index that contains overflow pages because duplicate key values are present, the result set returned by the descending scan may not be in exact reverse order of the result set that is returned with an ascending scan. The specified key values are returned in order, but the order of the

rows for the identical keys on the overflow pages may be different. For an explanation of how overflow pages in clustered indexes are stored, see the *Performance and Tuning Guide.*

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	Specifying new column headings in the **order by** clause of a select statement when the **union** operator is used is a Transact-SQL extension.

See Also

Commands	**compute Clause**, **declare**, **group by and having Clauses**, **select**, **where Clause**
System procedures	**sp_configure**, **sp_helpsort**, **sp_lock**, **sp_sysmon**

prepare transaction

Function

Used by DB-Library in a two-phase commit application to see if a server is prepared to commit a transaction.

Syntax

```
prepare tran[saction]
```

Comments

- For more information, see the *Open Client DB-Library Reference Manual*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

See Also

Commands	**begin transaction**, **commit**, **rollback**, **save transaction**

print

Function

Prints a user-defined message on the user's screen.

Syntax

```
print
   {format_string | @local_variable |
   @@global_variable}
       [, arg_list]
```

Keywords and Options

format_string – can be either a variable or a string of characters. The maximum length of *format_string* is 255 bytes.

Format strings can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow *format_string* when the text of the message is sent to the client.

To allow reordering of the arguments when format strings are translated to a language with a different grammatical structure, the placeholders are numbered. A placeholder for an argument appears in this format: "%*nn*!"—a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation point (!). The integer represents the argument number in the string in the argument list. "%1!" is the first argument in the original version, "%2!" is the second argument, and so on.

Indicating the position of the argument in this way makes it possible to translate correctly, even when the order in which the arguments appear in the target language is different.

For example, assume the following is an English message:

```
%1! is not allowed in %2!.
```

The German version of this message is:

```
%1! ist in %2! nicht zulassig.
```

The Japanese version of this message is:

%2! の中で %1! は許されません。

In this example, "%1!" represents the same argument in all three languages, as does "%2!". This example shows the reordering of

the arguments that is sometimes necessary in the translated form.

@local_variable – must be of type *char*, *nchar*, *varchar*, or *nvarchar*, and must be declared within the batch or procedure in which it is used.

@@global_variable – must be of type *char* or *varchar*, or be automatically convertible to these types, such as *@@version*. Currently, *@@version* is the only character-type global variable.

arg_list – may be a series of either variables or constants separated by commas. *arg_list* is optional unless a format string containing placeholders of the form "*%nn*!" is provided. In that case, the *arg_list* must have at least as many arguments as the highest numbered placeholder. An argument can be any datatype except *text* or *image*; it is converted to a character datatype before being included in the final message.

Examples

1.
```
if exists (select postalcode from authors
where postalcode = '94705')
print "Berkeley author"
```

   Prints "Berkeley author" if any authors in the *authors* table live in the 94705 ZIP code.

2.
```
declare @msg char(50)
select @msg = "What's up, doc?"
print @msg
```

```
What's up, doc?
```

   Declares a variable, assigns a value to the variable, and prints the value.

3.
```
declare @tabname varchar(30)
select @tabname = "titles"

declare @username varchar(30)
select @username = "ezekiel"

print "The table '%1!' is not owned by the user
'%2!'.", @tabname, @username
```

```
The table 'titles' is not owned
by the user 'ezekiel.'
```

   Demonstrates the use of variables and placeholders in messages.

Comments

- The maximum output string length of *format_string* plus all arguments after substitution is 512 bytes.
- If you use placeholders in a format string, keep this in mind: for each placeholder *n* in the string, the placeholders 1 through *n*-1 must also exist in the same string, although they do not have to be in numerical order. For example, you cannot have placeholders 1 and 3 in a format string without having placeholder 2 in the same string. If you omit a number in a format string, an error message is generated when **print** is executed.
- The *arg_list* must include an argument for each placeholder in the *format_string*, or the transaction is aborted. It is permissible to have more arguments than placeholders.
- To include a literal percent sign as part of the error message, use two percent signs ("%%") in the *format_string*. If you include a single percent sign ("%") in the *format_string* that is not used as a placeholder, Adaptive Server returns an error message.
- If an argument evaluates to NULL, it is converted into a zero-length character string. If you do not want zero-length strings in the output, use the **isnull** function. For example, if *@arg* is null, the following:

  ```
  declare @arg varchar(30)
  select @arg = isnull(col1, "nothing") from
  table_a where ...

  print "I think we have %1! here", @arg
  ```

  prints:

  ```
  I think we have nothing here.
  ```

- User-defined messages can be added to the system table *sysusermessages* for use by any application. Use **sp_addmessage** to add messages to *sysusermessages*; use **sp_getmessage** to retrieve messages for use by **print** and **raiserror**.
- Use **raiserror** instead of **print** if you want to print a user-defined error message and have the error number stored in *@@error*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

print permission defaults to all users. No permission is required to use it.

See Also

Commands	**declare, raiserror**
System procedures	**sp_addmessage, sp_getmessage**

quiesce database

Function

Suspends and resumes updates to a specified list of databases.

Syntax

quiesce database *tag_name* hold *dbname* [, *dbname*] ...

or

quiesce database *tag_name* release

Keywords and Options

tag_name – is a user-defined name that designates the list of databases to **hold** or **release**. The *tag_name* must conform to the rules for identifiers.

database – is the database name.

Examples

1. **quiesce database report_dbs hold salesdb, ordersdb**

   Suspends update activity on *salesdb* and *ordersdb*.

2. **quiesce database report_dbs release**

   Resumes update activity on the databases labeled *report_dbs*.

Comments

- Suspend updates to the *master* database **after** you have suspended updates to other user devices. If *master* is in a suspended state, attempts to suspend additional user databases generate warning messages.
- If the *master* database is in a suspended state, you should release *master* to perform updates before releasing any other user databases. If you attempt to release user databases while updates to *master* are suspended, Adaptive Server displays warning messages.
- **quiesce database** used with the **hold** keyword suspends all updates to the specified database. Transactions cannot update data in suspended databases, and background tasks such as the checkpoint process and housekeeper process will skip all databases that are in the suspended state.

- **quiesce database** used with the **release** keyword allows updates to resume on databases that were previously suspended.
- The **quiesce database hold** and **release** commands need not be executed from the same user session.
- If the databases specified in the **quiesce database hold** command contain distributed or multi-database transactions that are in the prepared state, Adaptive Server waits during a five second timeout period for those transactions to complete. If the transactions do not complete during the timeout period, **quiesce database hold** fails.
- You must perform a **dump database** command on each database at least once before you can suspend updates to those databases.
- If Adaptive Server is executing a **dump database** or **dump transaction** command on a database specified in **quiesce database hold**, the database is suspended only after the **dump** command completes.
- If you execute a **dump database** or **dump transaction** command on a database while updates to the database are suspended, Adaptive Server blocks those commands until the database is released with **quiesce database release**.
- You can specify a maximum of eight databases in a single **quiesce database hold** command. If you must suspend updates to additional databases, execute additional **quiesce database hold** commands.

Permissions

quiesce database permission defaults to System Administrators.

See Also

Commands	**dump database, dump transaction**
System procedures	**sp_helpdb, sp_who**

raiserror

Function

Prints a user-defined error message on the user's screen and sets a system flag to record that an error condition has occurred.

Syntax

```
raiserror error_number
   [{format_string | @local_variable}] [, arg_list]
   [with errordata restricted_select_list]
```

Keywords and Options

error_number – is a local variable or an integer with a value greater than 17,000. If the *error_number* is between 17,000 and 19,999, and *format_string* is missing or empty (““), Adaptive Server retrieves error message text from the *sysmessages* table in the *master* database. These error messages are used chiefly by system procedures.

If *error_number* is 20,000 or greater and *format_string* is missing or empty, **raiserror** retrieves the message text from the *sysusermessages* table in the database from which the query or stored procedure originates. Adaptive Server attempts to retrieve messages from either *sysmessages* or *sysusermessages* in the language defined by the current setting of *@@langid*.

format_string – is a string of characters with a maximum length of 255 bytes. Optionally, you can declare *format_string* in a local variable and use that variable with **raiserror** (see *@local_variable*).

raiserror recognizes placeholders in the character string that is to be printed out. Format strings can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow *format_string*, when the text of the message is sent to the client.

To allow reordering of the arguments, when format strings are translated to a language with a different grammatical structure, the placeholders are numbered. A placeholder for an argument appears in this format: “*%nn*!”—a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation point (!). The integer represents the argument number in the string in the argument list. “%1!” is the first argument in the original version, “%2!” is the second argument, and so on.

Indicating the position of the argument in this way makes it possible to translate correctly, even when the order in which the arguments appear in the target language is different from their order in the source language.

For example, assume the following is an English message:

```
%1! is not allowed in %2!.
```

The German version of this message is:

```
%1! ist in %2! nicht zulassig.
```

The Japanese version of this message is:

```
%2! の中で %1! は許されません。
```

In this example, "%1!" represents the same argument in all three languages, as does "%2!". This example shows the reordering of the arguments that is sometimes necessary in the translated form.

@local_variable – is a local variable containing the *format_string* value. It must be of type *char* or *varchar* and must be declared within the batch or procedure in which it is used.

arg_list – is a series of variables or constants separated by commas. *arg_list* is optional unless a format string containing placeholders of the form "*%nn*!" is provided. An argument can be any datatype except *text* or *image*; it is converted to the *char* datatype before being included in the final string.

If an argument evaluates to NULL, Adaptive Server converts it to a zero-length *char* string.

with errordata – supplies extended error data for Client-Library™ programs.

restricted_select_list – consists of one or more of the following items:

- "*", representing all columns in **create table** order.
- A list of column names in the order in which you want to see them. When selecting an existing IDENTITY column, you can substitute the **syb_identity** keyword, qualified by the table name, where necessary, for the actual column name.
- A specification to add a new IDENTITY column to the result table:

  column_name **= identity(*precision*)**

- A replacement for the default column heading (the column name), in the form:

  column_heading = column_name

  or:

  column_name column_heading

  or:

  column_name* as *column_heading

  The column heading may be enclosed in quotation marks for any of these forms. The heading must be enclosed in quotation marks if it is not a valid identifier (that is, if it is a reserved word, if it begins with a special character, or if it contains spaces or punctuation marks).

- An expression (a column name, constant, function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery).
- A built-in function or an aggregate
- Any combination of the items listed above

The *restricted_select_list* can also perform variable assignment, in the form:

```
@variable = expression
    [, @variable = expression ...]
```

Restrictions to *restricted_select_list* are:

- You cannot combine variable assignment with any of the other *restricted_select_list* options.
- You cannot use **from**, **where**, or other **select** clauses in *restricted_select_list*.
- You cannot use "*" to represent all columns in *restricted_select_list*.

For more information, see the *Transact-SQL User's Guide*.

Examples

1.
```
create procedure showtable_sp @tabname varchar(18)
as
if not exists (select name from sysobjects
    where name = @tabname)
    begin
      raiserror 99999 "Table %1! not found.",
        @tabname
    end
else
    begin
      select sysobjects.name, type, crdate, indid
      from sysindexes, sysobjects
      where sysobjects.name = @tabname
      and sysobjects.id = sysindexes.id
    end
```

   This stored procedure example returns an error if it does not find the table supplied with the *@tabname* parameter.

2.
```
sp_addmessage 25001,
"There is already a remote user named '%1!'
for remote server '%2!'."

raiserror 25001, jane, myserver
```

   This example adds a message to *sysusermessages*, then tests the message with **raiserror**, providing the substitution arguments.

3.
```
raiserror 20100 "Login must be at least 5
    characters long" with errordata "column" =
       "login", "server" = @@servername
```

   This example uses the **with errordata** option to return the extended error data *column* and *server* to a client application, to indicate which column was involved and which server was used.

Comments

- User-defined messages can be generated ad hoc, as in examples 1 and 3 above, or they can be added to the system table *sysusermessages* for use by any application, as shown in example 2. Use **sp_addmessage** to add messages to *sysusermessages*; use **sp_getmessage** to retrieve messages for use by **print** and **raiserror.**
- Error numbers for user-defined error messages must be greater than 20,000. The maximum value is 2,147,483,647 (2^{31} -1).
- The severity level of all user-defined error messages is 16. This level indicates that the user has made a a non-fatal error.

- The maximum output string length of *format_string* plus all arguments after substitution is 512 bytes.
- If you use placeholders in a format string, keep this in mind: for each placeholder *n* in the string, the placeholders *1* through *n-1* must exist in the same string, although they do not have to be in numerical order. For example, you cannot have placeholders 1 and 3 in a format string without having placeholder 2 in the same string. If you omit a number in a format string, an error message is generated when **raiserror** is executed.
- If there are too few arguments relative to the number of placeholders in *format_string*, an error message displays and the transaction is aborted. It is permissible to have more arguments than placeholders in *format_string*.
- To include a literal percent sign as part of the error message, use two percent signs ("%%") in the *format_string*. If you include a single percent sign ("%") in the *format_string* that is not used as a placeholder, Adaptive Server returns an error message.
- If an argument evaluates to NULL, it is converted into a zero-length *char* string. If you do not want zero-length strings in the output, use the **isnull** function.
- When **raiserror** is executed, the error number is placed in the global variable *@@error*, which stores the error number that was most recently generated by the system.
- Use **raiserror** instead of **print** if you want an error number stored in *@@error*.
- To include an *arg_list* with **raiserror**, put a comma after *error_number* or *format_string* before the first argument. To include extended error data, separate the first *extended_value* from *error_number*, *format_string*, or *arg_list* using a space (not a comma).

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

raiserror permission defaults to all users. No permission is required to use it.

See Also

Commands	**declare, print**
System procedures	**sp_addmessage, sp_getmessage**

readtext

Function

Reads *text* and *image* values, starting from a specified offset and reading a specified number of bytes or characters; if used with **readpast**, skips rows that have exclusive locks on them, without waiting and without generating a message.

Syntax

```
readtext [[database.]owner.]table_name.column_name
   text_pointer offset size
   [holdlock | noholdlock] [readpast]
   [using {bytes | chars | characters}]
   [at isolation {
       [ read uncommitted | 0 ] |
       [ read committed | 1 ] |
       [ repeatable read | 2 ]|
       [ serializable | 3 ] } ]
```

Keywords and Options

table_name.column_name – is the name of the *text* or *image* column You must include the table name. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

text_pointer – is a *varbinary*(16) value that stores the pointer to the *text* or *image* data. Use the **textptr** function to determine this value (see example 1). *text* and *image* data is not stored in the same set of linked pages as other table columns. It is stored in a separate set of linked pages. A pointer to the actual location is stored with the data; **textptr** returns this pointer.

offset – specifies the number of bytes or characters to skip before starting to read *text* or *image* data.

size – specifies the number of bytes or characters of data to read.

holdlock – causes the text value to be locked for reads until the end of the transaction. Other users can read the value, but they cannot modify it.

noholdlock – prevents the server from holding any locks acquired during the execution of this statement, regardless of the transaction isolation level currently in effect. You cannot specify both a **holdlock** and a **noholdlock** option in a query.

readpast – specifies that **readtext** should silently skip rows with exclusive locks, without waiting and without generating a message.

using – specifies whether **readtext** interprets the *offset* and *size* parameters as a number of bytes (**bytes**) or as a number of **textptr** characters (**chars** or **characters** are synonymous). This option has no effect when used with a single-byte character set or with *image* values (**readtext** reads *image* values byte by byte). If the **using** option is not given, **readtext** interprets the *size* and *offset* arguments as bytes.

at isolation – specifies the isolation level (0, 1, or 3) of the query. If you omit this clause, the query uses the isolation level of the session in which it executes (isolation level 1 by default). If you specify **holdlock** in a query that also specifies **at isolation read uncommitted**, Adaptive Server issues a warning and ignores the **at isolation** clause. For the other isolation levels, **holdlock** takes precedence over the **at isolation** clause.

read uncommitted – specifies isolation level 0 for the query. You can specify 0 instead of **read uncommitted** with the **at isolation** clause.

read committed – specifies isolation level 1 for the query. You can specify "1" instead of **read committed** with the **at isolation** clause.

repeatable read – specifies isolation level 2 for the query. You can specify "2" instead of **serializable** with the **at isolation** clause.

serializable – specifies isolation level 3 for the query. You can specify "3" instead of **serializable** with the **at isolation** clause.

Examples

1.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
where au_id =   "648-92-1872"
readtext blurbs.copy @val 1 5 using chars
```

   Selects the second through the sixth character of the *copy* column.

```
2. declare @val varbinary(16)
   select @val = textptr(copy) from blurbs readpast
   where au_id = "648-92-1872"
   readtext blurbs.copy @val 1 5 readpast using chars
```

Comments

- The **textptr** function returns a 16-byte binary string (text pointer) to the *text* or *image* column in the specified row or to the *text* or *image* column in the last row returned by the query, if more than one row is returned. It is best to declare a local variable to hold the text pointer, then use the variable with **readtext**.
- The value in the global variable *@@textsize*, which is the limit on the number of bytes of data to be returned, supersedes the size specified for **readtext** if it is less than that size. Use **set textsize** to change the value of *@@textsize*.
- When using bytes as the offset and size, Adaptive Server may find partial characters at the beginning or end of the *text* data to be returned. If it does, and character set conversion is on, the server replaces each partial character with a question mark (?) before returning the text to the client.
- Adaptive Server has to determine the number of bytes to send to the client in response to a **readtext** command. When the *offset* and *size* are in bytes, determining the number of bytes in the returned text is simple. When the offset and size are in characters, the server must take an extra step to calculate the number of bytes being returned to the client. As a result, performance may be slower when using characters as the *offset* and *size*. The **using characters** option is useful only when Adaptive Server is using a multibyte character set: this option ensures that **readtext** will not return partial characters.
- You cannot use **readtext** on *text* and *image* columns in views.
- If you attempt to use **readtext** on *text* values after changing to a multibyte character set, and you have not run **dbcc fix_text**, the command fails, and an error message instructs you to run **dbcc fix_text** on the table.

Using the readpast Option

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.

- The **readpast** option is incompatible with the **holdlock** option. If both are specified in a command, an error is generated and the command terminates.
- If the **readtext** command specifies **at isolation read uncommitted**, the **readpast** option generates a warning, but does not terminate the command.
- If the statement isolation level is set to 3, the **readpast** option generates an error and terminates the command.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored.
- If the session-wide isolation level is 0, the **readpast** option generates a warning, but does not terminate the command.

Standards and Compliance

Standard	Compliance level
SQL92	Transact-SQL extension

Permissions

readtext requires **select** permission on the table. **readtext** permission is transferred when **select** permission is transferred.

See Also

Commands	**set, writetext**
Datatypes	text and image Datatypes

reconfigure

Function

The **reconfigure** command currently has no effect; it is included to allow existing scripts to run without modification. In previous releases, **reconfigure** was required after the system procedure **sp_configure** to implement new configuration parameter settings.

Syntax

```
reconfigure
```

Comments

➤ ***Note***

If you have scripts that include **reconfigure**, change them at your earliest convenience. Although **reconfigure** is included in this release, it may not be supported in subsequent releases.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

reconfigure permission defaults to System Administrators and is not transferable.

See Also

System procedures	**sp_configure**

remove java

Function

Removes one or more Java-SQL classes, packages, or JARs from a database.

Use when Java classes are installed in the database. Refer to *Java in Adaptive Server Enterprise* for more information.

Syntax

```
remove java
   class class_name [, class_name]...
   | package package_name [, package_name]...
   | jar jar_name [, jar_name]...[retain classes]
```

Parameters

class *class_name* – the name of one or more Java classes to be removed from the database. The classes must be installed in the current database.

package *package_name* – the name of one or more Java packages to be removed. The packages must be stored in the current database.

jar *jar_name* – either a SQL identifier or character string value of up to 30 bytes that contains a valid SQL identifier.

Each *jar_name* must be equal to the name of a retained jar in the current database.

retain classes – specifies that the named JARs are no longer retained in the database, and the retained classes have no associated JAR.

Comments

- If a **remove java** statement is contained in a stored procedure, the current database is the database that is current when the procedure is created, not the database that is current when the procedure is called.

  If a **remove java** statement is not contained in a stored procedure, the current database is the database that is current when the **remove** statement is executed.
- If **class** or **package** is specified and any removed class has an associated JAR, then an exception is raised.

- If any stored procedure, table, or view contains a reference to a removed class as the datatype of a column, variable, or parameter, then an exception is raised.
- All removed classes are
  - Deleted from the current database.
  - Unloaded from the Java Virtual Machine (Java VM) of the current connection. The removed classes are not unloaded from the Java VMs of other connections.
- If any exception is raised during the execution of **remove java**, then all actions of **remove java** are cancelled.

Locks

- When you use **remove java**, an exclusive table lock is placed on *sysxtypes*.
- If **jar** is specified, then an exclusive table lock is placed on *sysjars*.

Permissions

You must be a System Administrator or Database Owner to use **remove java**.

See Also

System procedures	**sp_helpjava**
System tables	*sysjars*, *sysxtypes*
Utilities	**extractjava, installjava**

reorg

Function

Reclaims unused space on pages, removes row forwarding, or rewrites all rows in the table to new pages, depending on the option used.

Syntax

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]
reorg forwarded_rows tablename
   [with {resume,time = no_of_minutes}]
reorg compact tablename
   [with {resume, time = no_of_minutes}]
reorg rebuild [ tablename | indexname ]
```

Parameters and Keywords

reclaim_space – reclaims unused space left by deletes and updates. For each data page in a table, if there is unused space resulting from committed deletes or row-shortening updates, **reorg reclaim_space** rewrites the current rows contiguously, leaving all unused space at the end of the page. If there are no rows on the page, the page is deallocated.

tablename – specifies the name of the table to be reorganized. If *indexname* is specified, only the index is reorganized.

indexname – specifies the name of the index to be reorganized.

with resume – initiates reorganization from the point at which a previous **reorg** command terminated. Used when the previous **reorg** command specified a time limit (**time =** *no_of_minutes*).

with time = *no_of_minutes* – specifies the number of minutes that the **reorg** command is to run.

forwarded_rows – removes row forwarding.

compact – combines the functions of **reorg reclaim_space** and **reorg forwarded_rows** to both reclaim space and undo row forwarding in the same pass.

rebuild – if a table name is specified, rewrites all rows in a table to new pages, so that the table is arranged according to its clustered index (if one exists), with all pages conforming to current space management settings and with no forwarded rows and no gaps between rows on a page. If an index name is specified, **reorg** rebuilds that index while leaving the table accessible for read and update activities.

Examples

1. `reorg reclaim_space titles`

   Reclaims unused page space in the **titles** table.

2. `reorg reclaim_space titles titleind`

   Reclaims unused page space in the index **titleind**.

3. `reorg compact titles with time = 120`

   Initiates **reorg compact** on the *titles* table. **reorg** starts at the beginning of the table and continues for 120 minutes. If the **reorg** completes within the time limit, it returns to the beginning of the table and continues until the full time period has elapsed.

4. `reorg compact titles with resume, time = 30`

   Initiates **reorg compact** at the point where the previous **reorg compact** stopped and continues for 30 minutes.

Comments

- The table specified in the **reorg** command must have a datarows or datapages locking scheme.
- You cannot issue the **reorg** command within a transaction.
- **reorg rebuild** requires that you set the database option **select into/bulkcopy/pllsort** to **true** and run **checkpoint** in the database.
- **reorg rebuild** requires additional disk space equal to the size of the table and its indexes. You can find out how much space a table currently occupies by using **sp_spaceused**. You can use **sp_helpsegment** to check the amount of space available.
- After running **reorg rebuild**, you must dump the database before you can dump the transaction log.
- For more information, see the *System Administration Guide*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Administrator or the object owner to issue the **reorg** command.

See Also

System procedures	sp_chgattribute

return

Function

Exits from a batch or procedure unconditionally and provides an optional return status. Statements following **return** are not executed.

Syntax

```
return [integer_expression] [plan "abstract plan"]
```

Keywords and Options

integer_expression – is the integer value returned by the procedure. Stored procedures can return an integer value to a calling procedure or an application program.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan specified in the abstract plan language. Plans can only be specified for optimizable SQL statements, that is, queries that access tables. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples

1.
```
create procedure findrules @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
  return
end
else
begin
   select sysobjects.name, sysobjects.id,
   sysobjects.uid
 from sysobjects, master..syslogins
   where master..syslogins.name = @nm
   and sysobjects.uid = master..syslogins.suid
   and sysobjects.type = "R"
end
```

If no user name is given as a parameter, the **return** command causes the procedure to exit after a message has been sent to the user's screen. If a user name is given, the names of the rules created by that user in the current database are retrieved from the appropriate system tables.

```
2. print "Begin update batch"
   update titles
       set price = price + $3
       where title_id = 'BU2075'
   update titles
       set price = price + $3
       where title_id = 'BU1111'
   if (select avg(price) from titles
           where title_id like 'BU%') > $15
   begin
         print "Batch stopped; average price over $15"
       return
   end
   update titles
       set price = price + $2
         where title_id = 'BU1032'
```

If the updates cause the average price of business titles to exceed $15, the **return** command terminates the batch before any more updates are performed on *titles*.

```
3. create proc checkcontract @param varchar(11)
   as
   declare @status int
   if (select contract from titles where title_id =
   @param) = 1
      return 1
   else
      return 2
```

This procedure creates two user-defined status codes: a value of 1 is returned if the *contract* column contains a 1; a value of 2 is returned for any other condition (for example, a value of 0 on *contract* or a *title_id* that did not match a row).

Comments

- The return status value can be used in subsequent statements in the batch or procedure that executed the current procedure, but must be given in the form:

  execute *@retval* = *procedure_name*

  See **execute** for more information.

- Adaptive Server reserves 0 to indicate a successful return, and negative values in the range -1 to -99 to indicate different reasons for failure. If no user-defined return value is provided, the Adaptive Server value is used. User-defined return status values

must not conflict with those reserved by Adaptive Server. Numbers 0 and -1 to -14 are currently in use:

Table 6-30: Adaptive Server error return values

Value	Meaning
0	Procedure executed without error
-1	Missing object
-2	Datatype error
-3	Process was chosen as deadlock victim
-4	Permission error
-5	Syntax error
-6	Miscellaneous user error
-7	Resource error, such as out of space
-8	Non-fatal internal problem
-9	System limit was reached
-10	Fatal internal inconsistency
-11	Fatal internal inconsistency
-12	Table or index is corrupt
-13	Database is corrupt
-14	Hardware error

Values -15 to -99 are reserved for future Adaptive Server use.

- If more than one error occurs during execution, the status with the highest absolute value is returned. User-defined return values always take precedence over Adaptive Server-supplied return values.
- The **return** command can be used at any point where you want to exit from a batch or procedure. Return is immediate and complete: statements after **return** are not executed.
- A stored procedure cannot return a NULL return status. If a procedure attempts to return a null value, for example, using **return** *@status* where *@status* is NULL, a warning message is generated, and a value in the range of 0 to -14 is returned.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

return permission defaults to all users. No permission is required to use it.

See Also

Commands	**begin...end, execute, if...else, while**

revoke

Function

Revokes permissions or roles from users or roles.

Syntax

To revoke permission to access database objects:

```
revoke [grant option for]
   {all [privileges] | permission_list}
   on { table_name [(column_list)]
       | view_name [(column_list)]
       | stored_procedure_name}
   from {public | name_list | role_name}
   [cascade]
```

To revoke permission to create database objects, execute **set proxy**, or execute **set session authorization**:

```
revoke {all [privileges] | command_list}
   from {public | name_list | role_name}
```

To revoke a role from a user or another role:

```
revoke role {role_name [, role_name ...]} from
   {grantee [, grantee ...]}
```

Keywords and Options

all – when used to revoke permission to access database objects (the first syntax format), **all** revokes all permissions applicable to the specified object. All object owners can use **revoke all** with an object name to revoke permissions on their own objects.

Only the System Administrator or the Database Owner can revoke permission to revoke **create** command permissions (the second syntax format). When used by the System Administrator, **revoke all** revokes all **create** permissions (**create database**, **create default**, **create procedure**, **create rule**, **create table**, and **create view**). When the Database Owner uses **revoke all**, Adaptive Server revokes all **create** permissions except **create database**, and prints an informational message.

all does not apply to **set proxy** or **set session authorization**.

permission_list – is a list of permissions to revoke. If more than one permission is listed, separate them with commas. The following

table illustrates the access permissions that can be granted and revoked on each type of object:

Object	*permission_list* Can Include:
Table	**select, insert, delete, update, references**
View	**select, insert, delete, update**
Column	**select, update, references** Column names can be specified in either *permission_list* or *column_list* (see example 2).
Stored procedure	**execute**

Permissions can be revoked only by the user who granted them.

command_list – is a list of commands. If more than one command is listed, separate them with commas. The command list can include **create database**, **create default**, **create procedure**, **create rule**, **create table**, **create view**, **set proxy**, or **set session authorization**. **create database** permission can be revoked only by a System Administrator and only from within the *master* database.

set proxy and **set session authorization** are identical; the only difference is that **set session authorization** follows the SQL standard, and **set proxy** is a Transact-SQL extension. Revoking permission to execute **set proxy** or **set session authorization** revokes permission to become another user in the server. Permissions for **set proxy** or **set session authorization** can be revoked only by a System Security Officer, and only from within the *master* database.

table_name – is the name of the table on which you are revoking permissions. The table must be in your current database. Only one object can be listed for each **revoke** statement.

column_list – is a list of columns, separated by commas, to which the privileges apply. If columns are specified, only **select** and **update** permissions can be revoked.

view_name – is the name of the view on which you are revoking permissions. The view must be in your current database. Only one object can be listed for each **revoke** statement.

stored _procedure_name – is the name of the stored procedure on which you are revoking permissions. The stored procedure must

be in your current database. Only one object can be listed for each **revoke** statement.

public – is all users. For object access permissions, **public** excludes the object owner. For object creation permissions or **set proxy** authorizations, **public** excludes the Database Owner. You cannot **grant** permissions **with grant option** to "public" or to other groups or roles.

name_list – is a list of user and / or group names, separated by commas.

role – is the name of a system or user-defined role. Use **revoke role** to revoke granted roles from roles or users.

role_name – is the name of a system or user-defined role. This allows you to revoke permissions from all users who have been granted a specific role. The role name can be either a system role or a user-defined role created by a System Security Officer with **create role**. Either type of role can be granted to a user with the **grant role** command. In addition, the system procedure **sp_role** can be used to grant system roles.

grantee – is the name of a system role, user-defined role, or a user, from whom you are revoking a role.

grant option for – revokes **with grant option** permissions, so that the user(s) specified in *name_list* can no longer grant the specified permissions to other users. If those users have granted permissions to other users, you must use the **cascade** option to revoke permissions from those users. The user specified in *name_list* retains permission to access the object, but can no longer grant access to other users. **grant option for** applies only to object access permissions, not to object creation permissions.

cascade – revokes the specified object access permissions from all users to whom the revokee granted permissions. Applies only to object access permissions, not to object creation permissions. (When you use **revoke** without **grant option for**, permissions granted to other users by the revokee are also revoked: the cascade occurs automatically.)

Examples

1. ```
   revoke insert, delete
   on titles
   from mary, sales
   ```

   Revokes **insert** and **delete** permissions on the *titles* table from Mary and the "sales" group.

2. ```
   revoke update
   on titles (price, advance)
   from public
   ```

   or:

   ```
   revoke update (price, advance)
   on titles
   from public
   ```

   Two ways to revoke **update** permission on the *price* and *advance* columns of the *titles* table from "public".

3. ```
   revoke create database, create table
   from mary, john
   ```

   Revokes permission from Mary and John to use the **create database** and **create table** commands. Because **create database** permission is being revoked, this command must be executed by a System Administrator from within the *master* database. Mary and John's **create table** permission will be revoked only within the *master* database.

4. ```
   revoke set proxy from harry, billy
   ```

   Revokes permission from Harry and Billy to execute either **set proxy** or **set session authorization** to impersonate another user in the server.

5. ```
   revoke set session authorization from sso_role
   ```

   Revokes permission from users with **sso_role** to execute either **set proxy** or **set session authorization**.

6. ```
   revoke set proxy from vip_role
   ```

   Revokes permission from users with **vip_role** to impersonate another user in the server. **vip_role** must be a role defined by a System Security Officer with the **create role** command.

7. ```
   revoke all
   from mary
   ```

   Revokes all object creation permissions from Mary in the current database.
   ```

8.
```
revoke all
on titles
from mary
```

   Revokes all object access permissions on the *titles* table from Mary.

9.
```
revoke references
on titles (price, advance)
from tom
```

   or:

```
revoke references (price, advance)
on titles
from tom
```

   Two ways to revoke Tom's permission to create a referential integrity constraint on another table that refers to the *price* and *advance* columns in the *titles* table.

10.
```
revoke execute on new_sproc
from oper_role
```

   Revokes permission to execute the stored procedure *new_sproc* from all users who have been granted the Operator role.

11.
```
revoke grant option for
insert, update, delete
on authors
from john
cascade
```

   Revokes John's permission to grant **insert, update**, and **delete** permissions on the *authors* table to other users. Also revokes from other users any such permissions that John has granted.

12.
```
revoke role doctor_role from specialist_role
```

   Revokes doctor_role from specialist_role.

13.
```
revoke role doctor_role, surgeon_role from
specialist_role, intern_role, mary, tom
```

   Revokes "doctor_role" and "surgeon_role" from "specialist_role" and "intern_role", and from users Mary and Tom.

Comments

- See the **grant** command for more information about permissions.
- You can revoke permissions only on objects in your current database.

- You can only revoke permissions that were granted by you.
- You cannot revoke a role from a user while the user is logged in.
- **grant** and **revoke** commands are order sensitive. When there is a conflict, the command issued most recently takes effect.
- The word **to** can be substituted for the word **from** in the **revoke** syntax.
- If you do not specify **grant option for** in a **revoke** statement, **with grant option** permissions are revoked from the user along with the specified object access permissions. In addition, if the user has granted the specified permissions to any other users, all of those permissions are revoked. In other words, the **revoke** cascades.
- **revoke grant option** revokes the user's ability to grant the specified permission to other users, but does not revoke the permission itself from that user. If the user has granted that permission to others, you must use the **cascade** option; otherwise, you will receive an error message and the **revoke** will fail.

  For example, say you revoke the **with grant option** permissions from the user Bob on *titles*, with this statement:

  ```
  revoke grant option for select
  on titles
  from bob
  cascade
  ```

  - If Bob has not granted this permission to other users, this command revokes his ability to do so, but he retains **select** permission on the *titles* table.
  - If Bob has granted this permission to other users, you must use the **cascade** option. If you do not, you will receive an error message and the **revoke** will fail. **cascade** revokes this **select** permission from all users to whom Bob has granted it, as well as their ability to grant it to others.
- A **grant** statement adds one row to the *sysprotects* system table for each user, group, or role that receives the permission. If you subsequently **revoke** the permission from the user or group, Adaptive Server removes the row from *sysprotects*. If you revoke the permission from only selected group members, but not from the entire group to which it was granted, Adaptive Server retains the original row and adds a new row for the revoke.
- Permission to issue the **create trigger** command is granted to users by default. When you revoke permission for a user to create triggers, a revoke row is added in the *sysprotects* table for that

user. To grant permission to issue **create trigger**, you must issue two grant commands. The first command removes the revoke row from *sysprotects*; the second inserts a grant row. If you revoke permission to create triggers, the user cannot create triggers even on tables that the user owns. Revoking permission to create triggers from a user affects only the database where the revoke command was issued.

Revoking *set proxy* and *set session authorization*

- To revoke **set proxy** or **set session authorization** permission, or to revoke roles, you must be a System Security Officer, and you must be in the *master* database.
- **set proxy** and **set session authorization** are identical, with one exception: **set session authorization** follows the SQL standard. If you are concerned about using only SQL standard commands and syntax, use **set session authorization**.
- **revoke all** does **not** include **set proxy** or **set session authorization** permissions.

Revoking from Roles, Users and Groups

- Permissions granted to roles override permissions granted to individual users or groups. Therefore, if you revoke a permission from a user who has been granted a role, and the role has that same permission, the user will retain it. For example, say John has been granted the System Security Officer role, and **sso_role** has been granted permission on the *sales* table. If John's individual permission on *sales* is revoked, he can still access *sales* because his role permissions override his individual permissions.
- Revoking a specific permission from "public" or from a group also revokes it from users who were individually granted the permission.
- Database user groups allow you to **grant** or **revoke** permissions to more than one user at a time. A user is always a member of the default group, "public" and can be a member of only one other group. Adaptive Server's installation script assigns a set of permissions to "public."

  Create groups with the system procedure **sp_addgroup** and remove groups with **sp_dropgroup**. Add new users to a group with **sp_adduser**. Change a user's group membership with **sp_changegroup**. To display the members of a group, use **sp_helpgroup**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Database Object Access Permissions

revoke permission for database objects defaults to object owners. An object owner can revoke permission from other users on his or her own database objects.

Command Execution Permissions

Only a System Administrator can revoke **create database** permission, and only from the *master* database. Only a System Security Officer can revoke **create trigger** permission.

Proxy and Session Authorization Permissions

Only a System Security Officer can revoke **set proxy** or **set session authorization**, and only from the *master* database.

Role Permissions

You can revoke roles only from the *master* database. Only a System Security Officer can revoke **sso_role**, **oper_role** or a user-defined role from a user or a role. Only System Administrators can revoke **sa_role** from a user or a role. Only a user who has both **sa_role** and **sso_role** can revoke a role which includes **sa_role**.

See Also

Commands	**grant, setuser, set**
Functions	**proc_role**
System procedures	**sp_activeroles**, **sp_adduser**, **sp_changedbowner**, **sp_changegroup**, **sp_displaylogin**, **sp_displayroles**, **sp_dropgroup**, **sp_dropuser**, **sp_helpgroup**, **sp_helprotect**, **sp_helpuser**, **sp_modifylogin**, **sp_role**

rollback

Function

Rolls back a user-defined transaction to the named savepoint in the transaction or to the beginning of the transaction.

Syntax

```
rollback {tran[saction] | work}
   [transaction_name | savepoint_name]
```

Keywords and Options

transaction | **tran** | **work** – is optional.

transaction_name – is the name assigned to the outermost transaction. It must conform to the rules for identifiers.

savepoint_name – is the name assigned to the savepoint in the **save transaction** statement. The name must conform to the rules for identifiers.

Examples

```
1. begin transaction
   delete from publishers where pub_id = "9906"
   rollback transaction
```

Rolls back the transaction.

Comments

- **rollback transaction** without a *transaction_name* or *savepoint_name* rolls back a user-defined transaction to the beginning of the outermost transaction.
- **rollback transaction** *transaction_name* rolls back a user-defined transaction to the beginning of the named transaction. Though you can nest transactions, you can roll back only the outermost transaction.
- **rollback transaction** *savepoint_name* rolls a user-defined transaction back to the matching **save transaction** *savepoint_name*.

Restrictions

- If no transaction is currently active, the **commit** or **rollback** statement has no effect.

- The **rollback** command must appear within a transaction. You cannot roll back a transaction after **commit** has been entered.

Rolling Back an Entire Transaction

- **rollback** without a savepoint name cancels an entire transaction. All the transaction's statements or procedures are undone.
- If no *savepoint_name* or *transaction_name* is given with the **rollback** command, the transaction is rolled back to the first **begin transaction** in the batch. This also includes transactions that were started with an implicit **begin transaction** using the chained transaction mode.

Rolling Back to a Savepoint

- To cancel part of a transaction, use **rollback** with a *savepoint_name*. A savepoint is a marker set within a transaction by the user with the command **save transaction**. All statements or procedures between the savepoint and the **rollback** are undone.

  After a transaction is rolled back to a savepoint, it can proceed to completion (executing any SQL statements after that **rollback**) using **commit**, or it can be canceled altogether using **rollback** without a savepoint. There is no limit on the number of savepoints within a transaction.

Rollbacks Within Triggers and Stored Procedures

- In triggers or stored procedures, **rollback** statements without transaction or savepoint names roll back all statements to the first explicit or implicit **begin transaction** in the batch that called the procedure or fired the trigger.
- When a trigger contains a **rollback** command without a savepoint name, the rollback aborts the entire batch. Any statements in the batch following the rollback are not executed.
- A remote procedure call (RPC) is executed independently from any transaction in which it is included. In a standard transaction (that is, not using Open Client™ DB-Library two-phase commit), commands executed via an RPC by a remote server are not rolled back with **rollback** and do not depend on **commit** to be executed.
- For complete information on using transaction management statements and on the effects of **rollback** on stored procedures, triggers, and batches, see the *Transact-SQL User's Guide*.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The **rollback transaction** and **rollback tran** forms of the statement and the use of a transaction name are Transact-SQL extensions.

Permissions

rollback permission defaults to "public." No permission is required to use it.

See Also

Commands	**begin transaction, commit, create trigger, save transaction**

rollback trigger

Function

Rolls back the work done in a trigger, including the data modification that caused the trigger to fire, and issues an optional **raiserror** statement.

Syntax

```
rollback trigger
   [with raiserror_statement]
```

Keywords and Options

with *raiserror_statement* – specifies a **raiserror** statement, which prints a user-defined error message and sets a system flag to record that an error condition has occurred. This provides the ability to raise an error to the client when the rollback trigger is executed so that the transaction state in the error reflects the rollback. For information about the syntax and rules defining *raiserror_statement*, see the **raiserror** command.

Examples

```
1. rollback trigger with raiserror 25002
   "title_id does not exist in titles table."
```

Rolls back a trigger and issues the user-defined error message 25002.

Comments

- When **rollback trigger** is executed, Adaptive Server aborts the currently executing command and halts execution of the rest of the trigger.
- If the trigger that issues **rollback trigger** is nested within other triggers, Adaptive Server rolls back all work done in these triggers up to and including the update that caused the first trigger to fire.
- Adaptive Server ignores a **rollback trigger** statement that is executed outside a trigger and does not issue a **raiserror** associated with the statement. However, a **rollback trigger** statement executed outside a trigger but inside a transaction generates an error that causes Adaptive Server to roll back the transaction and abort the current statement batch.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

rollback trigger permission defaults to "public." No permission is required to use it.

See Also

Commands	**create trigger, raiserror, rollback**

save transaction

Function

Sets a savepoint within a transaction.

Syntax

```
save transaction savepoint_name
```

Keywords and Options

savepoint_name – is the name assigned to the savepoint. It must conform to the rules for identifiers.

Examples

```
1. begin transaction royalty_change

   update titleauthor
   set royaltyper = 65
   from titleauthor, titles
   where royaltyper = 75
   and titleauthor.title_id = titles.title_id
   and title = "The Gourmet Microwave"

   update titleauthor
   set royaltyper = 35
   from titleauthor, titles
   where royaltyper = 25
   and titleauthor.title_id = titles.title_id
   and title = "The Gourmet Microwave"

   save transaction percentchanged

   update titles
   set price = price * 1.1
```

```
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

rollback transaction percentchanged

commit transaction
```

After updating the *royaltyper* entries for the two authors, insert the savepoint *percentchanged*, then determine how a 10 percent increase in the book's price would affect the authors' royalty earnings. The transaction is rolled back to the savepoint with the **rollback transaction** command.

Comments

- For complete information on using transaction statements, see the *Transact-SQL User's Guide*.
- A savepoint is a user-defined marker within a transaction that allows portions of a transaction to be rolled back. The command **rollback** *savepoint_name* rolls back to the indicated savepoint; all statements or procedures between the savepoint and the **rollback** are undone.

  Statements preceding the savepoint are not undone—but neither are they committed. After rolling back to the savepoint, the transaction continues to execute statements. A **rollback** without a savepoint cancels the entire transaction. A **commit** allows it to proceed to completion.
- If you nest transactions, **save transaction** creates a savepoint only in the outermost transaction.
- There is no limit on the number of savepoints within a transaction.
- If no *savepoint_name* or t*ransaction_name* is given with the **rollback** command, all statements back to the first **begin transaction** in a batch are rolled back, and the entire transaction is canceled.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

save transaction permission defaults to "public." No permission is required to use it.

See Also

Commands	**begin transaction, commit, rollback**

select

Function

Retrieves rows from database objects.

Syntax

select ::=
select [**all** | **distinct**] *select_list*
[*into_clause*]
[*from_clause*]
[*where_clause*]
[*having_clause*]
[*order_by_clause*]
[*compute_clause*]
[*read_only_clause*]
[*isolation_clause*]
[*browse_clause*]
[*plan_clause*]

select_list ::= Defined under "Keywords and Options," below

into_clause **::=**
into [[*database*.]*owner*.]t*able_name*
[**lock** {**datarows** | **datapages** | **allpages** }]
[**with** *into_option* [*, into_option*] *...*]

into_option ::=
| **max_rows_per_page =** *num_rows*
| **exp_row_size** = *num_bytes*
| **reservepagegap** = *num_pages*
| **identity_gap** = gap

from_clause **::= from** *table_reference* [,*table_reference*]...

table_reference **:=**
table_view_name
| *ANSI_join*

table_view_name **:=**
[[*database*.]*owner*.] {{*table_name* |*view_name*} [as] [*correlation_name*]
[**index** {*index_name* | *table_name* }]
[**parallel** [*degree_of_parallelism*]]
[**prefetch size**][**lru** | **mru**]}]}
[**holdlock** | **noholdlock**] [**readpast**] [**shared**]

ANSI_join **=**
table_reference join_type **join** *table_reference join_conditions*

join_type = inner | left [outer] | right [outer]

join_conditions **= on** *search_conditions*

where_clause ::= **where** *search_conditions*

group_by_clause ::=
group by [**all**] *aggregate_free_expression*
[, *aggregate_free_expression*]...

having_clause ::= **having** *search_conditions*

order_by_clause ::=
order by sort_clause [, sort_clause]...

sort_clause ::=
{ [[[*database*.]*owner*.]{*table_name*.|*view_name*.}]*column_name*
| *select_list_number*
| *expression* }
[**asc** | **desc**]

compute_clause ::=
compute *row_aggregate*(*column_name*)
[, *row_aggregate*(*column_name*)]...
[**by** *column_name* [, *column_name*]...]

read_only_clause ::=
for {**read only** | **update** [**of** *column_name_list*]}

isolation_clause ::=
at isolation
{ **read uncommitted** | **0** }
| { **read committed** | **1** }
| { **repeatable read** | **2** }
| { **serializable** | **3** }

browse_clause ::= **for browse**

plan_clause ::= **plan** "**abstract plan**"

Keywords and Options

all – includes all rows in the results. **all** is the default.

distinct – includes only unique rows in the results. **distinct** must be the first word in the select list. **distinct** is ignored in browse mode.

Null values are considered equal for the purposes of the keyword **distinct**: only one NULL is selected, no matter how many are encountered.

Even when configured for case-insensitive sort order, **distinct** reports “smith” and “Smith” as two distinct rows.

select_list – consists of one or more of the following items:

- “*”, representing all columns in **create table** order.

- A list of column names in the order in which you want to see them. When selecting an existing IDENTITY column, you can substitute the **syb_identity** keyword, qualified by the table name, where necessary, for the actual column name.
- A specification to add a new IDENTITY column to the result table:

  ***column_name* = identity(*precision*)**
- A replacement for the default column heading (the column name), in the form:

  column_heading = column_name

  or:

  column_name column_heading

  or:

  column_name* as *column_heading

  The column heading can be enclosed in quotation marks for any of these forms. The heading must be enclosed in quotation marks if it is not a valid identifier (that is, if it is a reserved word, if it begins with a special character, or if it contains spaces or punctuation marks).
- An expression (a column name, constant, function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery)
- A built-in function or an aggregate
- Any combination of the items listed above

  The *select_list* can also assign values to variables, in the form:

  ```
  @variable = expression
      [, @variable = expression ...]
  ```

  You cannot combine variable assignment with any other *select_list* option.

into – creates a new table based on the columns specified in the select list and the rows chosen in the **where** clause. See "Using select into" in this section.

lock datarows | datapages | allpages – specifies the locking scheme to be used for a table created with a **select into** command. The default is the server-wide setting for the configuration parameter **lock scheme**.

max_rows_per_page – limits the number of rows on data pages for a table created with **select into**. Unlike **fillfactor**, the **max_rows_per_page** value is maintained when data is inserted or deleted. **max_rows_per_page** is not supported on data-only-locked tables.

exp_row_size = *num_bytes* – specifies the expected row size for a table created with the **select into** command. Valid only for datarows and datapages locking schemes and only for tables that have variable-length rows. Valid values are 0, 1, and any value greater than the minimum row length and less than the maximum row length for the table. The default value is 0, which means that a server-wide default is used.

reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages that is to be left as **select into** allocates extents to store data. This option is valid only for the **select into** command. For each specified *num_pages*, one empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

readpast – specifies that the query should silently skip rows with exclusive locks, without waiting and without generating a message.

from – indicates which tables and views to use in the **select** statement. It is required except when the select list contains no column names (that is, it contains constants and arithmetic expressions only):

```
select 5 x, 2 y, "the product is", 5*2 Result
x           y                         Result
----------- ----------- -------------- -----------
          5           2 the product is          10
```

At most, a query can reference 16 tables and 12 worktables (such as those created by aggregate functions). The 16-table limit includes:

- Tables (or views on tables) listed in the **from** clause
- Each instance of multiple references to the same table (self-joins)
- Tables referenced in subqueries
- Tables being created with **into**
- Base tables referenced by the views listed in the **from** clause

view_name, *table_name* – lists tables and views used in the **select** statement. Specify the database name if the table or view is in

another database, and specify the owner's name if more than one table or view of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

If there is more than one table or view in the list, separate their names by commas. The order of the tables and views following the keyword **from** does not affect the results.

You can query tables in different databases in the same statement.

Table names and view names can be given correlation names (aliases), either for clarity or to distinguish the different roles that tables or views play in self-joins or subqueries. To assign a correlation name, give the table or view name, then a space, then the correlation name, like this:

```
select pub_name, title_id
    from publishers pu, titles t
    where t.pub_id = pu.pub_id
```

All other references to that table or view (for example in a **where** clause) must use the correlation name. Correlation names cannot begin with a numeral.

index *index_name* – specifies the index to use to access *table_name*. You cannot use this option when you select from a view, but you can use it as part of a **select** clause in a **create view** statement.

parallel – specifies a parallel partition or index scan, if Adaptive Server is configured to allow parallel processing.

degree_of_parallelism – specifies the number of worker processes that will scan the table or index in parallel. If set to 1, the query executes serially.

prefetch *size* – specifies the I/O size, in kilobytes, for tables bound to caches with large I/Os configured. Valid values for size are 2, 4, 8, and 16. You cannot use this option when you select from a view, but you can use it as part of a **select** clause in a **create view** statement. The procedure **sp_helpcache** shows the valid sizes for the cache an object is bound to or for the default cache.

If Component Integration Services is enabled, you cannot use **prefetch** for remote servers.

lru | mru – specifies the buffer replacement strategy to use for the table. Use **lru** to force the optimizer to read the table into the cache on the

MRU / LRU (most recently used / least recently used) chain. Use **mru** to discard the buffer from cache and replace it with the next buffer for the table. You cannot use this option when you select from a view, but you can use it as part of a **select** clause in a **create view** statement.

holdlock – makes a shared lock on a specified table or view more restrictive by holding it until the transaction completes (instead of releasing the shared lock as soon as the required data page is no longer needed, whether or not the transaction has completed).

The **holdlock** option applies only to the table or view for which it is specified, and only for the duration of the transaction defined by the statement in which it is used. Setting the **transaction isolation level 3** option of the **set** command implicitly applies a **holdlock** for each **select** statement within a transaction. The keyword **holdlock** is not permitted in a **select** statement that includes the **for browse** option. You cannot specify both a **holdlock** and a **noholdlock** option in a query.

If Component Integration Services is enabled, you cannot use **holdlock** for remote servers.

noholdlock – prevents the server from holding any locks acquired during the execution of this **select** statement, regardless of the transaction isolation level currently in effect. You cannot specify both a **holdlock** and a **noholdlock** option in a query.

shared – instructs Adaptive Server to use a shared lock (instead of an update lock) on a specified table or view. This allows other clients to obtain an update lock on that table or view. You can use the **shared** keyword only with a **select** clause included as part of a **declare cursor** statement. For example:

```
declare shared_crsr cursor
for select title, title_id
from titles shared
where title_id like "BU%"
```

You can use the **holdlock** keyword in conjunction with **shared** after each table or view name, but **holdlock** must precede **shared**.

ANSI join – an inner or outer join that uses the ANSI syntax. The **from** clause specifies which tables are to be joined.

inner – includes only the rows of the inner and outer tables that meet the conditions of the **on** clause. The result set of a query that includes an inner join does not include any null supplied rows for

the rows of the outer table that do not meet the conditions of the **on** clause.

outer – includes all the rows from the outer table whether or not they meet the conditions of the **on** clause. If a row does not meet the conditions of the **on** clause, values from the inner table are stored in the joined table as null values. The **where** clause of an ANSI outer join restricts the rows that are included in the query result.

left – left joins retain all the rows of the table reference listed on the left of the join clause. The left table reference is referred to as the outer table or row-preserving table.

In the queries below, *T1* is the outer table and *T2* is the inner table:

```
T1 left join T2
T2 right join T1
```

right – right joins retain all the rows of the table reference on the right of the join clause (see example above).

search_conditions – used to set the conditions for the rows that are retrieved. A search condition can include column names, expressions, arithmetic operators, comparison operators, the keywords **not**, **like**, **is null**, **and**, **or**, **between**, **in**, **exists**, **any**, and **all**, subqueries, case expressions, or any combination of these items. See "where Clause" for more information.

group by – finds a value for each group. These values appear as new columns in the results, rather than as new rows.

When **group by** is used with standard SQL, each item in the select list must either have a fixed value in every row in the group or be used with aggregate functions, which produce a single value for each group. Transact-SQL has no such restrictions on the items in the select list. Also, Transact-SQL allows you to group by any expression (except by a column alias); with standard SQL, you can group by a column only.

You can use the aggregates listed in Table 6-31 with **group by** (*expression* is almost always a column name):

Table 6-31: Results of using aggregates with group by

Aggregate Function	Result
sum([all | distinct] *expression)*	Total of the values in the numeric column.
avg([all | distinct] *expression)*	Average of the values in the numeric column.
count([all | distinct] *expression)*	Number of (distinct) non-null values in the column.
count(*)	Number of selected rows.
max(*expression)*	Highest value in the column.
min*(expression)*	Lowest value in the column.

See "group by and having Clauses" for more information.

A table can be grouped by any combination of columns—that is, groups can be nested within each other. You cannot group by a column heading; you must use a column name, an expression, or a number representing the position of the item in the select list.

group by all – includes all groups in the results, even those that do not have any rows that meet the search conditions (see "group by and having Clauses" for an example).

aggregate_free_expression – is an expression that includes no aggregates.

having – sets conditions for the **group by** clause, similar to the way that **where** sets conditions for the **select** clause. There is no limit on the number of conditions that can be included.

You can use a **having** clause without a **group by** clause.

If any columns in the select list do not have aggregate functions applied to them and are not included in the query's **group by** clause (illegal in standard SQL), the meanings of **having** and **where** are somewhat different.

In this situation, a **where** clause restricts the rows that are included in the calculation of the aggregate, but does not restrict the rows returned by the query. Conversely, a **having** clause restricts the rows returned by the query, but does not affect the

calculation of the aggregate. See "group by and having Clauses" for examples.

order by – sorts the results by columns. In Transact-SQL, you can use **order by** for items that do not appear in the select list. You can sort by a column name, a column heading (or alias), an expression, or a number representing the position of the item in the **select list** (the *select_list_number*). If you sort by select list number, the columns to which the **order by** clause refers must be included in the select list, and the select list cannot be * (asterisk).

asc – sorts results in ascending order (the default).

desc – sorts results in descending order.

compute – used with row aggregates (**sum**, **avg**, **min**, **max**, and **count**) to generate control break summary values. The summary values appear as additional rows in the query results, allowing you to see detail and summary rows with one statement.

You cannot use a **select into** clause with **compute**.

If you use **compute by**, you must also use an **order by** clause. The columns listed after **compute by** must be identical to or a subset of those listed after **order by**, and must be in the same left-to-right order, start with the same expression, and not skip any expressions.

For example, if the **order by** clause is:

```
order by a, b, c
```

the **compute by** clause can be any (or all) of these:

```
compute by a, b, c
compute by a, b
compute by a
```

The keyword **compute** can be used without **by** to generate grand totals, grand counts, and so on. **order by** is optional if you use **compute** without **by**. See "compute Clause" for details and examples.

If Component Integration Services is enabled, you cannot use **compute** for remote servers.

for {**read only** | **update**} – specifies that a cursor result set is read-only or updatable. You can use this option only within a stored procedure and only when the procedure defines a query for a cursor. In this case, the **select** is the only statement allowed in the procedure. It defines the **for read only** or **for update** option (instead of

the **declare cursor** statement). This method of declaring cursors provides the advantage of page-level locking while fetching rows.

If the **select** statement in the stored procedure is not used to define a cursor, Adaptive Server ignores the **for read only | update** option. See the Embedded SQL™ documentation for more information about using stored procedures to declare cursors. For information about read-only or updatable cursors, see the *Transact-SQL User's Guide*.

of *column_name_list* – is the list of columns from a cursor result set defined as updatable with the **for update** option.

at isolation – specifies the isolation level (0, 1, 2 or 3) of the query. If you omit this clause, the query uses the isolation level of the session in which it executes (isolation level 1 by default). The **at isolation** clause is valid only for single queries or within the **declare cursor** statement. Adaptive Server returns a syntax error if you use **at isolation**:

- With a query using the **into** clause
- Within a subquery
- With a query in the **create view** statement
- With a query in the **insert** statement
- With a query using the **for browse** clause

If there is a **union** operator in the query, you must specify the **at isolation** clause after the last **select**. If you specify **holdlock**, **noholdlock**, or **shared** in a query that also specifies **at isolation read uncommitted**, Adaptive Server issues a warning and ignores the **at isolation** clause. For the other isolation levels, **holdlock** takes precedence over the **at isolation** clause. For more information about isolation levels, see the *Transact-SQL User's Guide*.

If Component Integration Services is enabled, you cannot use **at isolation** for remote servers.

read uncommitted | 0 – specifies isolation level 0 for the query.

read committed | 1 – specifies isolation level 1 for the query.

repeatable read | 2 – specifies transaction isolation level 2 for the query.

serializable | 3 – specifies isolation level 3 for the query.

for browse – must be attached to the end of a SQL statement sent to Adaptive Server in a DB-Library browse application. See the *Open Client DB-Library Reference Manual* for details.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples

1. select * from publishers

```
pub_id pub_name                    city                 state
------ --------------------------- -------------------- -----
0736    New Age Books              Boston               MA
0877    Binnet & Hardley           Washington           DC
1389    Algodata Infosystems       Berkeley             CA
```

Selects all rows and columns from the *publishers* table.

2. select pub_id, pub_name, city, state from publishers

Selects all rows from specific columns of the *publishers* table.

3. select "The publisher's name is", Publisher = pub_name, pub_id from publishers

```
                    Publisher                      pub_id
----------------------- ------------------------------ ------
The publisher's name is New Age Books                  0736
The publisher's name is Binnet & Hardley               0877
The publisher's name is Algodata Infosystems           1389
```

Selects all rows from specific columns of the *publishers* table, substituting one column name and adding a string to the output.

4. select type as Type, price as Price from titles

Selects all rows from specific columns of the *titles* table, substituting column names.

5. select title_id, title, price into bus_titles lock datarows with reservepagegap = 10 from titles where type = "business"

Specifies the locking scheme and the reserve page gap for **select into**.

```
6. select title, price
   from titles readpast
       where type = "news"
       and price between $20 and $30
```

Selects only the rows that are not exclusively locked. If any other user has an exclusive lock on a qualifying row, that row is not returned.

```
7. select pub_id, total = sum (total_sales)
       into #advance_rpt
   from titles
   where advance < $10000
       and total_sales is not null
   group by pub_id
   having count(*) > 1
```

Selects specific columns and rows, placing the results into the temporary table *#advance_rpt*.

```
8. select "Author_name" = au_fname + " " + au_lname
       into #tempnames
       from authors
```

Concatenates two columns and places the results into the temporary table *#tempnames*.

```
9. select type, price, advance from titles
   order by type desc
   compute avg(price), sum(advance) by type
   compute sum(price), sum(advance)
```

Selects specific columns and rows, returns the results ordered by type from highest to lowest, and calculates summary information.

```
10.select type, price, advance from titles
   compute sum(price), sum(advance)
```

Selects specific columns and rows, and calculates totals for the *price* and *advance* columns.

```
11.select * into coffeetabletitles from titles
   where price > $20
```

Creates the *coffeetabletitles* table, a copy of the *titles* table which includes only books priced over $20.

```
12.select * into newtitles from titles
   where 1 = 0
```

Creates the *newtitles* table, an empty copy of the *titles* table.

```
13.select title_id, title
       from titles (index title_id_ind prefetch 16)
       where title_id like "BU%"
```

Gives an optimizer hint.

```
14.select sales_east.syb_identity,
   sales_west.syb_identity
   from sales_east, sales_west
```

Selects the IDENTITY column from the *sales_east* and *sales_west* tables by using the *syb_identity* keyword.

```
15.select *, row_id = identity(10)
   into newtitles from titles
```

Creates the *newtitles* table, a copy of the *titles* table with an IDENTITY column.

```
16.select pub_id, pub_name
   from publishers
   at isolation read uncommitted
```

Specifies a transaction isolation level for the query.

```
17.begin tran
   select type, avg(price)
       from titles
       group by type
   at isolation repeatable read
```

Selects from *titles* using the **repeatable read** isolation level. No other user can change values in or delete the affected rows until the transaction completes.

```
18.select ord_num from salesdetail
       (index salesdetail parallel 3)
```

Gives an optimizer hint for the parallel degree for the query.

```
19.select au_id, titles.title_id, title, price
   from titleauthor inner join titles
   on titleauthor.title_id = titles.title_id
   and price > 15
```

Joins the *titleauthor* and the *titles* tables on their *title_id* columns. The result set only includes those rows that contain a *price* greater than 15.

```
20.select au_fname, au_lname, pub_name
   from authors left join publishers
   on authors.city = publishers.city
```

The result set contains all the authors from the *authors* table. The authors who do not live in the same city as their publishers produce null values in the *pub_name* column. Only the authors who live in the same city as their publishers, Cheryl Carson and Abraham Bennet, produce a non-null value in the *pub_name* column.

Comments

- The keywords in the **select** statement, as in all other statements, must be used in the order shown in the syntax statement.
- The keyword **all** can be used after **select** for compatibility with other implementations of SQL. **all** is the default. Used in this context, **all** is the opposite of **distinct**. All retrieved rows are included in the results, whether or not some are duplicates.
- Except in **create table**, **create view**, and **select into** statements, column headings may include any characters, including blanks and Adaptive Server keywords, if the column heading is enclosed in quotes. If the heading is not enclosed in quotes, it must conform to the rules for identifiers.
- Column headings in **create table**, **create view**, and **select into** statements, as well as table aliases, must conform to the rules for identifiers.
- To insert data with **select** from a table that has null values in some fields into a table that does not allow null values, you must provide a substitute value for any NULL entries in the original table. For example, to insert data into an *advances* table that does not allow null values, this example substitutes "0" for the NULL fields:

  ```
  insert advances
  select pub_id, isnull(advance, 0) from titles
  ```

  Without the **isnull** function, this command would insert all the rows with non-null values into the *advances* table, and produce error messages for all rows where the *advance* column in the *titles* table contained NULL.

  If you cannot make this kind of substitution for your data, you cannot insert data containing null values into the columns with the NOT NULL specification.

Two tables can be identically structured, and yet be different as to whether null values are permitted in some fields. Use **sp_help** to see the null types of the columns in your table.

- The default length of the *text* or *image* data returned with a **select** statement is 32K. Use **set textsize** to change the value. The size for the current session is stored in the global variable *@@textsize*. Certain client software may issue a **set textsize** command on logging into Adaptive Server.
- Data from remote Adaptive Servers can be retrieved through the use of remote procedure calls. See **create procedure** and **execute** for more information.
- A **select** statement used in a cursor definition (through **declare cursor**) must contain a **from** clause, but it cannot contain a **compute**, **for browse**, or **into** clause. If the **select** statement contains any of the following constructs, the cursor is considered read-only and not updatable:
  - **distinct** option
  - **group by** clause
  - Aggregate functions
  - **union** operator

  If you declare a cursor inside a stored procedure with a **select** statement that contains an **order by** clause, that cursor is also considered read-only. Even if it is considered updatable, you cannot delete a row using a cursor that is defined by a **select** statement containing a join of two or more tables. See **declare cursor** for more information.
- If a **select** statement that assigns a value to a variable returns more than one row, the last returned value is assigned to the variable. For example:

```
declare @x varchar(40)
select @x = pub_name from publishers
print @x
```

```
(3 rows affected)
Algodata Infosystems
```

Using ANSI join syntax

- Before you write queries using the ANSI inner and outer join syntax, make sure you read “Outer Joins” in Chapter 4, “Joins:

Retrieving Data From Several Tables", in the *Transact-SQL User's Guide*.

Using *select into*

- **select into** is a two-step operation. The first step creates the new table, and the second step inserts the specified rows into the new table.

  Because the rows inserted by **select into** operations are not logged, **select into** commands cannot be issued within user-defined transactions, even if the **ddl in tran** database option is set to **true**. Page allocations during **select into** operations are logged, so large **select into** operations may fill the transaction log.

  If a **select into** statement fails after creating a new table, Adaptive Server does **not** automatically drop the table or deallocate its first data page. This means that any rows inserted on the first page before the error occurred remain on the page. Check the value of the *@@error* global variable after a **select into** statement to be sure that no error occurred. Use the **drop table** statement to remove the new table, then reissue the **select into** statement.

- The name of the new table must be unique in the database and must conform to the rules for identifiers. You can also **select into** temporary tables (see examples 7, 8, and 11).

- You cannot **select into** a table that already exists. Instead, use the **insert...select** command. See **insert** for more information.

- Any rules, constraints, or defaults associated with the base table are not carried over to the new table. Bind rules or defaults to the new table using **sp_bindrule** and **sp_bindefault.**

- **select into** does not carry over the base table's **max_rows_per_page** value, and it creates the new table with a **max_rows_per_page** value of 0. Use **sp_chgattribute** to set the **max_rows_per_page** value.

- The **select into/bulkcopy/pllsort** option must be set to **true** (by executing **sp_dboption**) in order to **select into** a permanent table. You do not have to set the **select into/bulkcopy/pllsort** option to **true** in order to **select into** a temporary table, since the temporary database is never recovered.

  After you have used **select into** in a database, you must perform a full database dump before you can use the **dump transaction** command. **select into** operations log only page allocations and not changes to data rows. Therefore, changes are not recoverable from transaction logs. In this situation, issuing the **dump transaction**

statement produces an error message instructing you to use **dump database** instead.

By default, the **select into/bulkcopy/pllsort** option is set to **false** in newly created databases. To change the default situation, set this option to **true** in the *model* database.

- **select into** runs more slowly while a **dump database** is taking place.
- You can use **select into** to create a duplicate table with no data by having a false condition in the **where** clause (see example 12).
- You must provide a column heading for any column in the select list that contains an aggregate function or any expression. The use of any constant, arithmetic or character expression, built-in functions, or concatenation in the select list requires a column heading for the affected item. The column heading must be a valid identifier or must be enclosed in quotation marks (see examples 7 and 8).
- Because functions allow null values, any column in the select list that contains a function other than **convert** or **isnull** allows null values.
- You cannot use **select into** inside a user-defined transaction or in the same statement as a **compute** clause.
- To select an IDENTITY column into a result table, include the column name (or the **syb_identity** keyword) in the **select** statement's *column_list*. The new column observes the following rules:
  - If an IDENTITY column is selected more than once, it is defined as NOT NULL in the new table. It does not inherit the IDENTITY property.
  - If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as NULL if any column in the expression allows nulls; otherwise, it is created as NOT NULL.
  - If the **select** statement contains a **group by** clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created NULL; others are NOT NULL.
  - An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a NULL column, the new column is defined as NULL. Otherwise, it is defined as NOT NULL.

- You cannot use **select into** to create a new table with multiple IDENTITY columns. If the **select** statement includes both an existing IDENTITY column and a new IDENTITY specification of the form *column_name* = **identity***(precision)*, the statement fails.
- If Component Integration Services is enabled, and if the **into** table resides on Adaptive Server, Adaptive Server uses bulk copy routines to copy the data into the new table. Before doing a **select into** with remote tables, set the **select into/bulkcopy** database option to **true**.
- For information about the Embedded SQL command **select into** *host_var_list*, see the *Open Client Embedded SQL Reference Manual*.

Converting the Null Properties of a Target Column with *select...into*

- Use the **convert** command to change the nullability of a target column into which you are selecting data. For example, the following selects data from the *titles* table into a target table named *temp_titles*, but converts the *total_sales* column from **null** to **not null**:

```
select title, convert (char(100) not null,
total_sales) into #tempsales
from titles
```

Specifying a Lock Scheme with *select...into*

- The **lock** option, used with **select...into**, allows you to specify the locking scheme for the table created by the command. If you do not specify a locking scheme, the default locking scheme, as set by the configuration parameter **lock scheme**, is applied.
- When you use the **lock** option, you can also specify the space management properties **max_rows_per_page**, **exp_row_size**, and **reservepagegap**.

  You can change the space management properties for a table created with **select into**, using the **sp_chgattribute** system procedure.

Using *index*, *prefetch*, and *lru | mru*

- The **index**, **prefetch** and **lru | mru** options specify the index, cache and I/O strategies for query execution. These options override the choices made by the Adaptive Server optimizer. Use them with caution, and always check the performance impact with **set statistics io on**. For more information about using these options, see the *Performance and Tuning Guide*.

Using *parallel*

- The **parallel** option reduces the number of worker threads that the Adaptive Server optimizer can use for parallel processing. The *degree_of_parallelism* cannot be greater than the configured **max parallel degree**. If you specify a value that is greater than the configured **max parallel degree**, the optimizer ignores the **parallel** option.
- When multiple worker processes merge their results, the order of rows that Adaptive Server returns may vary from one execution to the next. To get rows from partitioned tables in a consistent order, use an **order** by clause, or override parallel query execution by using **parallel 1** in the **from** clause of the query.
- A **from** clause specifying **parallel** is ignored if any of the following conditions is true:
  - The select statement is used for an update or insert.
  - The **from** clause is used in the definition of a cursor.
  - **parallel** is used in the **from** clause within any inner query blocks of a subquery.
  - The select statement creates a view.
  - The table is the inner table of an outer join.
  - The query specifies **min** or **max** on the table and specifies an index.
  - An unpartitioned clustered index is specified or is the only **parallel** option.
  - The query specifies **exists** on the table.
  - The value for the configuration parameter **max scan parallel degree** is 1 and the query specifies an index.
  - A nonclustered index is covered. For information on index covering, see Chapter 4, “How Indexes Work,” in the *Performance and Tuning Guide*.
  - The table is a system table or a virtual table.
  - The query is processed using the OR strategy. For an explanation of the OR strategy, see the *Performance and Tuning Guide*.
  - The query will return a large number of rows to the user.

Using *readpast*

- The **readpast** option allows a **select** command to access the specified table without being blocked by incompatible locks held by other tasks. **readpast** queries can only be performed on data-only-locked tables.
- If the **readpast** option is specified for an allpages-locked table, the **readpast** option is ignored. The command operates at the isolation level specified for the command or session. If the isolation level is 0, dirty reads are performed, and the command returns values from locked rows and does not block. If the isolation level is 1 or 3, the command blocks when pages with incompatible locks must be read.
- The interactions of session-level isolation levels and **readpast** on a table in a **select** command are shown in Table 6-32.

Table 6-32: Effects of session-level isolation levels and readpast

Session Isolation Level	Effects
0, **read uncommitted** (dirty reads)	**readpast** is ignored, and rows containing uncommitted transactions are returned to the user. A warning message is printed.
1, **read committed**	Rows or pages with incompatible locks are skipped; no locks are held on the rows or pages read
2, **repeatable read**	Rows or pages with incompatible locks are skipped; shared locks are held on all rows or pages that are read until the end of the statement or transaction; holds locks on all pages read by the statement until the transaction completes.
3, **serializable**	**readpast** is ignored, and the command executes at level 3. The command blocks on any rows or pages with incompatible locks.

- **select** commands that specify **readpast** fail with an error message if they also include any of the following:
  - An **at isolation** clause, specifying **0** or **read uncommitted**
  - An **at isolation** clause, specifying **3** or **serializable**
  - The **holdlock** keyword on the same table

- If **at isolation 2** or **at isolation repeatable read** is specified in a **select** query that specifies **readpast**, shared locks are held on the **readpast** tables until the statement or transaction completes.
- If a **select** command with the **readpast** option encounters a text column that has an incompatible lock on it, **readpast** locking retrieves the row, but returns the text column with a value of **null**. No distinction is made, in this case, between a text column containing a null value and a null value returned because the column is locked.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The following are Transact-SQL extensions: • **select into** to create a new table • **lock** clauses • **compute** clauses • Global and local variables • **index** clause, **prefetch, parallel** and **lru | mru** • **holdlock**, **noholdlock**, and **shared** keywords • "*column_heading = column_name*" • Qualified table and column names • **select** in a **for browse** clause • The use, within the select list, of columns that are not in the **group by** list and have no aggregate functions • **at isolation repeatable read | 2** option

Permissions

select permission defaults to the owner of the table or view, who can transfer it to other users.

See Also

Commands	**compute Clause**, **create index**, **create trigger**, **delete**, **group by and having Clauses**, **insert**, **order by Clause**, **set**, **union Operator**, **update**, **where Clause**
Functions	**avg**, **count**, **isnull**, **max**, **min**, **sum**
System procedures	**sp_cachestrategy**, **sp_chgattribute**, **sp_dboption**

set

Function

Sets Adaptive Server query-processing options for the duration of the user's work session; sets some options inside a trigger or stored procedure; activates or deactivates a role in the current session; specifies the length of time that a query waits to acquire a lock before aborting and returning an error message; sets the transaction isolation level to isolation level 2, repeatable reads; specifies that the query should be optimized using simulated statistics; sets distributed transaction processing options.

Syntax

```
set ansinull {on | off}

set ansi_permissions {on | off}

set arithabort [arith_overflow | numeric_truncation]
   {on | off}

set arithignore [arith_overflow] {on | off}

set {chained, close on endtran, nocount, noexec,
   parseonly, procid, self_recursion, showplan,
   sort_resources} {on | off}

set char_convert {off | on [with {error | no_error}] |
   charset [with {error | no_error}]}

set cis_rpc_handling {on | off}

set [clientname client_name | clienthostname
   host_name | clientapplname application_name]

set cursor rows number for cursor_name

set {datefirst number, dateformat format,
   language language}

set fipsflagger {on | off}

set flushmessage {on | off}

set forceplan {on | off}

set identity_insert [database.[owner.]]table_name
   {on | off}

set jtc {on | off}

set lock { wait [ numsecs ] | nowait }

set offsets {select, from, order, compute, table,
   procedure, statement, param, execute} {on | off}
```

```
set parallel_degree number
set plan {dump | load } [group_name] {on | off}
set plan exists check {on | off}
set plan replace {on | off}
set prefetch [on|off]
set process_limit_action {abort | quiet | warning}
set proxy login_name
set quoted_identifier {on | off}
set role {"sa_role" | "sso_role" | "oper_role" |
   role_name [with passwd "password"]} {on | off}
set {rowcount number, textsize number}
set scan_parallel_degree number
set session authorization login_name
set sort_merge {on | off}
set statistics {io, subquerycache, time} {on | off}
set statistics simulate { on | off }
set strict_dtm_enforcement {on | off}
set string_rtruncation {on | off}
set table count number
set textsize {number}
set transaction isolation level {
   [ read uncommitted | 0 ] |
   [ read committed | 1 ] |
   [ repeatable read | 2 ]|
   [ serializable | 3 ] }
set transactional_rpc {on | off}
```

Keywords and Options

ansinull – determines whether or not evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons or aggregate functions, also called **set functions**, is compliant with the SQL92 standard. When you use **set ansinull on**, Adaptive Server generates a warning each time an aggregate function eliminates a null-valued operand from calculation. This option does not affect how Adaptive Server evaluates NULL values in other kinds of SQL statements such as **create table**.

The SQL standards requires that if either one of the two operands of an equality comparison is NULL, the result is UNKNOWN. Transact-SQL treats NULL values differently. If one of the operands is a column, parameter, or variable, and the other operand is the NULL constant or a parameter or variable whose value is NULL, the result is either TRUE or FALSE.

ansi_permissions – determines whether SQL92 permission requirements for **delete** and **update** statements are checked. The default is **off**. Table 6-33 summarizes permission requirements:

Table 6-33: Permissions required for update and delete

Command	Permissions Required with *set ansi_permissions off*	Permissions Required with *set ansi_permissions on*
update	• **update** permission on columns where values are being set	• **update** permission on columns where values are being set • **select** permission on all columns appearing in **where** clause • **select** permission on all columns on right side of **set** clause
delete	• **delete** permission on table	• **delete** permission on table • **select** permission on all columns appearing in **where** clause

arithabort – determines how Adaptive Server behaves when an arithmetic error occurs. The two **arithabort** options, **arithabort arith_overflow** and **arithabort numeric_truncation**, handle different types of arithmetic errors. You can set each option independently or set both options with a single **set arithabort on** or **set arithabort off** statement.

- **arithabort arith_overflow** specifies Adaptive Server's behavior following a divide-by-zero error or a loss of precision during an explicit or implicit datatype conversion. This type of error is serious. The default setting, **arithabort arith_overflow on**, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, **arithabort arith_overflow on** does not roll back earlier commands in the batch; however, Adaptive Server does not execute any statements in the batch that follow the error-generating statement.

  If you set **arithabort arith_overflow off**, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.

- **arithabort numeric_truncation** specifies Adaptive Server's behavior following a loss of scale by an exact numeric type during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, **arithabort numeric_truncation on**, aborts the statement that causes the error, but Adaptive Server continues to process other statements in the transaction or batch. If you set **arithabort numeric_truncation off**, Adaptive Server truncates the query results and continues processing.

arithignore arith_overflow – determines whether Adaptive Server displays a message after a divide-by-zero error or a loss of precision. By default, the **arithignore** option is set to **off**. This causes Adaptive Server to display a warning message after any query that results in numeric overflow. To have Adaptive Server ignore overflow errors, use **set arithignore on**. You can omit the optional **arith_overflow** keyword without any effect.

chained – begins a transaction just before the first data retrieval or data modification statement at the beginning of a session and after a transaction ends. In chained mode, Adaptive Server implicitly executes a **begin transaction** command before the following statements: **delete**, **fetch**, **insert**, **open**, **select**, and **update**. You cannot execute **set chained** within a transaction.

char_convert – enables or disables character set conversion between Adaptive Server and a client. If the client is using Open Client DB-Library release 4.6 or later, and the client and server use different character sets, conversion is turned on during the login process and is set to a default based on the character set the client is using. You can also use **set char_convert** *charset* to start conversion between the server character set and a different client character set.

charset can be either the character set's ID or a name from *syscharsets* with a *type* value of less than 2000.

set char_convert off turns conversion off so that characters are sent and received unchanged. **set char_convert on** turns conversion on if it is turned off. If character set conversion was not turned on during the login process or by the **set char_convert** command, **set char_convert on** generates an error message.

When the **with no_error** option is included, Adaptive Server does not notify an application when characters from Adaptive Server cannot be converted to the client's character set. Error reporting is initially turned on when a client connects with Adaptive

Server: if you do not want error reporting, you must turn it off for each session with **set char_convert {on | charset} with no_error.** To turn error reporting back on within a session, use **set char_convert {on | charset} with error.**

Whether or not error reporting is turned on, the bytes that cannot be converted are replaced with ASCII question marks (?).

See the *System Administration Guide* for a more complete discussion of error handling in character set conversion.

cis_rpc_handling – determines whether Component Integration Services handles outbound remote procedure call (RPC) requests by default.

clientapplname – assigns an application an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same application name. After you assign a new name to an application, it appears in the *sysprocesses* table under the new name.

clienthostname – assigns a host an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same host name. After you assign a new name to a host, it appears in the *sysprocesses* table under the new name.

clientname – assigns a client an individual name. This is useful for differentiating among clients in a system where many clients connect to Adaptive Server using the same client name. After you assign a new name to a user, they appear in the *sysprocesses* table under the new name.

close on endtran – causes Adaptive Server to close all cursors opened within a transaction at the end of that transaction. A transaction ends by the use of either the **commit** or **rollback** statement. However, only cursors declared within the scope that sets this option (stored procedure, trigger, and so on) are affected. For more information about cursor scopes, see the *Transact-SQL User's Guide.*

For more information about the evaluated configuration, see the *System Administration Guide.*

cursor rows – causes Adaptive Server to return the *number* of rows for each cursor **fetch** request from a client application. The *number* can be a numeric literal with no decimal point or a local variable of type *integer.* If the *number* is less than or equal to zero, the value is set to 1. You can set the **cursor rows** option for a cursor, whether it is

open or closed. However, this option does not affect a **fetch** request containing an **into** clause. *cursor_name* specifies the cursor for which to set the number of rows returned.

datefirst – sets the first week day to a number from 1 to 7. The us_english language default is 1 (Sunday).

dateformat – sets the order of the date parts *month/day/year* for entering *datetime* or *smalldatetime* data. Valid arguments are *mdy*, *dmy*, *ymd*, *ydm*, *myd*, and *dym*. The us_english language default is *mdy*.

fipsflagger – determines whether Adaptive Server displays a warning message when Transact-SQL extensions to entry level SQL92 are used. By default, Adaptive Server does not tell you when you use nonstandard SQL. This option does not disable SQL extensions. Processing completes when you issue the non-ANSI SQL command.

flushmessage – determines when Adaptive Server returns messages to the user. By default, messages are stored in a buffer until the query that generated them is completed or the buffer is filled to capacity. Use **set flushmessage on** to return messages to the user immediately, as they are generated.

forceplan – causes the query optimizer to use the order of the tables in the **from** clause of a query as the join order for the query plan. **forceplan** is generally used when the optimizer fails to choose a good plan. Forcing an incorrect plan can have severely bad effects on I/O and performance. For more information, see the *Performance and Tuning Guide*.

identity_insert – determines whether explicit inserts into a table's IDENTITY column are allowed. (Updates to an IDENTITY column are never allowed.) This option can be used only with base tables. It cannot be used with views or set within a trigger.

Setting **identity_insert** *table_name* **on** allows the table owner, Database Owner, or System Administrator to explicitly insert a value into an IDENTITY column. Inserting a value into the IDENTITY column allows you to specify a seed value for the column or to restore a row that was deleted in error. Unless you have created a unique index on the IDENTITY column, Adaptive Server does not verify the uniqueness of the inserted value; you can insert any positive integer.

The table owner, Database Owner, or System Administrator can use the **set identity_insert** *table_name* **on** command on a table with an

IDENTITY column in order to enable the manual insertion of a value into an IDENTITY column. However, only the following users can actually insert a value into an IDENTITY column, when **identity_insert** is **on**:

- Table owner
- Database Owner, if granted explicit **insert** permission on the column by the table owner
- Database Owner, impersonating the table owner by using the **setuser** command

Setting **identity_insert** *table_name* **off** restores the default behavior by prohibiting explicit inserts to IDENTITY columns. At any time, you can use **set identity_insert** *table_name* **on** for a single database table within a session.

ljtc – toggles join transitive closure. For more information, see the *Performance and Tuning Guide*.

language – is the official name of the language that displays system messages. The language must be installed on Adaptive Server. The default is us_english.

nocount – controls the display of rows affected by a statement. **set nocount on** disables the display of rows; **set nocount off** reenables the count of rows.

noexec – compiles each query but does not execute it. **noexec** is often used with **showplan**. After you set **noexec on**, no subsequent commands are executed (including other **set** commands) until you set **noexec off**.

lock wait – specifies the length of time that a command waits to acquire locks before aborting and returning an error.

numsecs – specifies the number of seconds a command is to wait to acquire a lock. Valid values are from 0 to 2147483647, the maximum value for an integer.

lock nowait – specifies that if a command cannot acquire a lock immediately, it returns an error and fails. **set lock nowait** is equivalent to **set lock wait 0**.

offsets – returns the position of specified keywords (with relation to the beginning of the query) in Transact-SQL statements. The keyword list is a comma-separated list that can include any of the following Transact-SQL constructs: **select**, **from**, **order**, **compute**, **table**,

procedure, **statement, param**, and **execute**. Adaptive Server returns offsets if there are no errors. This option is used in Open Client DB-Library only.

parallel_degree – specifies an upper limit for the number of worker processes used in the parallel execution of a query. This number must be less than or equal to the number of worker processes per query, as set by the **max parallel degree** configuration parameter. The *@@parallel_degree* global variable stores the current setting.

parseonly – checks the syntax of each query and returns any error messages without compiling or executing the query. Do not use **parseonly** inside a stored procedure or trigger.

plan – introduces an abstract plan command. For more information, see Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide*.

dump – enables or disables capturing abstract plans for the current connection. If a *group_name* is not specified, the plans are stored in the default group, *ap_stdout*.

load – enables or disables loading abstract plans for the current connection. If a *group_name* is not specified, the plans are loaded from the default group, *ap_stdin*.

group_name – is the name of the abstract plan group to use for loading or storing plans.

exists check – when used with **set plan load**, stores hash keys for up to 20 queries from an abstract plan group in a per-user cache.

replace – enables or disables replacing existing abstract plans during plan capture mode. By default, plan replacement is off.

prefetch – enables or disables large I/Os to the data cache.

process_limit_action – specifies whether Adaptive Server executes parallel queries when an insufficient number of worker processes are available. Under these circumstances, when **process_limit_action** is set to **quiet**, Adaptive Server silently adjusts the plan to use a degree of parallelism that does not exceed the number of available processes. If **process_limit_action** is set to **warning** when an insufficient number of worker processes are available, Adaptive Server issues a warning message when adjusting the plan; and if **process_limit_action** is set to **abort**, Adaptive Server aborts the query

and issues an explanatory message an insufficient number of worker processes are available.

procid – returns the ID number of the stored procedure to Open Client DB-Library / C (not to the user) before sending rows generated by the stored procedure.

proxy – allows you to assume the permissions, login name, and *suid* (server user ID) of *login_name*. For *login_name*, specify a valid login from *master..syslogins*, enclosed in quotation marks. To revert to your original login name and *suid*, use **set proxy** with your original *login_name*.

See "Using Proxies" for more information.

quoted_identifier – determines whether Adaptive Server recognizes delimited identifiers. By default, **quoted_identifier** is **off** and all identifiers must conform to the rules for valid identifiers. If you use **set quoted_identifier on**, you can use table, view, and column names that begin with a non-alphabetic character, include characters that would not otherwise be allowed, or are reserved words, by enclosing the identifiers within double quotation marks. Delimited identifiers cannot exceed 28 bytes, may not be recognized by all front-end products, and may produce unexpected results when used as parameters to system procedures.

When **quoted_identifier** is **on**, all character strings enclosed within double quotes are treated as identifiers. Use single quotes around character or binary strings.

role – turns the specified role on or off during the current session. When you log in, all system roles that have been granted to you are turned on. Use **set role *role_name* off** to turn a role off, and **set role *role_name* on** to turn it back on again, as needed. System roles are **sa_role**, **sso_role**, and **oper_role**. If you are not a user in the current database, and if there is no "guest" user, you cannot set **sa_role off**, because there is no server user ID for you to assume.

role_name – is the name of any user-defined role created by the System Security Officer. User-defined roles are not turned on by default. To set user-defined roles to activate at login, the user or the System Security Officer must use **set role on**.

with *passwd* – specifies the password to activate the role. If a user-defined role has an attached password, you must specify the password to activate the role.

rowcount – causes Adaptive Server to stop processing the query (**select**, **insert**, **update**, or **delete**) after the specified number of rows are affected. The *number* can be a numeric literal with no decimal point or a local variable of type *integer*. To turn this option off, use:

```
set rowcount 0
```

scan_parallel_degree – specifies the maximum session-specific degree of parallelism for hash-based scans (parallel index scans and parallel table scans on nonpartitioned tables). This number must be less than or equal to the current value of the **max scan parallel degree** configuration parameter. The *@@scan_parallel_degree* global variable stores the current setting.

self_recursion – determines whether Adaptive Server allows triggers to cause themselves to fire again (this is called **self-recursion**). By default, Adaptive Server does not allow self recursion in triggers. You can turn this option on only for the duration of a current client session; its effect is limited by the scope of the trigger that sets it. For example, if the trigger that sets **self_recursion on** returns or causes another trigger to fire, this option reverts to **off**. This option works only within a trigger and has no effect on user sessions.

session authorization – is identical to **set proxy**, with this exception: **set session authorization** follows the SQL standard, while **set proxy** is a Transact-SQL extension.

showplan – generates a description of the processing plan for the query. The results of **showplan** are of use in performance diagnostics. **showplan** does not print results when it is used inside a stored procedure or trigger. For parallel queries, **showplan** output also includes the adjusted query plan at run-time, if applicable. For more information, see the *Performance and Tuning Guide*.

sort_merge – enables or disables the use of sort-merge joins during a session. For more information, see the *Performance and Tuning Guide*.

sort_resources – generates a description of the sorting plan for a **create index** statement. The results of **sort_resources** are of use in determining whether a sort operation will be done serially or in parallel. When **sort_resouces** is **on**, Adaptive Server prints the sorting plan but does not execute the **create index** statement. For more information, see Chapter 13, "Parallel Sorting," in the *Performance and Tuning Guide*.

statistics io – displays the following statistics information for each table referenced in the statement:

- the number of times the table is accessed (scan count)
- the number of logical reads (pages accessed in memory)
- and the number of physical reads (database device accesses)

For each command, **statistics io** displays the number of buffers written.

If Adaptive Server has been configured to enforce resource limits, **statistics io** also displays the total I/O cost. For more information, see Chapter 16, "Using the set statistics Commands," in the *Performance and Tuning Guide*.

statistics subquerycache – displays the number of cache hits, misses, and the number of rows in the subquery cache for each subquery.

statistics time – displays the amount of time Adaptive Server used to parse and compile for each command. For each step of the command, **statistics time** displays the amount of time Adaptive Server used to execute the command. Times are given in milliseconds and timeticks, the exact value of which is machine-dependent.

statistics simulate – specifies that the optimizer should use simulated statistics to optimize the query.

strict_dtm_enforcement – determines whether the server will propagate transactions to servers that do not support Adaptive Server transaction coordination services. The default value is inherited from the value of the **strict dtm enforcement** configuration parameter.

string_rtruncation – determines whether Adaptive Server raises a SQLSTATE exception when an **insert** or **update** command truncates a *char* or *varchar* string. If the truncated characters consist only of spaces, no exception is raised. The default setting, **off**, does not raise the SQLSTATE exception, and the character string is silently truncated.

table count – sets the number of tables that Adaptive Server will consider at one time while optimizing a join. The default used depends on the number of tables in the join:.

Tables Joined	Tables Considered at a Time
2 – 25	4
26 – 37	3
38 - 50	2

Valid values are 0–8. A value of 0 resets the default behavior. A value greater than 8 defaults to 8. **table count** may improve the optimization of certain join queries, but it increases the compilation cost.

textsize – specifies the maximum size in bytes of *text* or *image* type data that is returned with a **select** statement. The *@@textsize* global variable stores the current setting. To reset **textsize** to the default size (32K), use the command:

```
set textsize 0
```

The default setting is 32K in **isql**. Some client software sets other default values.

transaction isolation level – sets the transaction isolation level for your session. After you set this option, any current or future transactions operate at that isolation level.

read uncommitted | 0 – Scans at isolation level 0 do not acquire any locks. Therefore, the result set of a level 0 scan may change while the scan is in progress. If the scan position is lost due to changes in the underlying table, a unique index is required to restart the scan. In the absence of a unique index, the scan may be aborted.

By default, a unique index is required for a level 0 scan on a table that does not reside in a read-only database. You can override this requirement by forcing the Adaptive Server to choose a nonunique index or a table scan, as follows:

```
select * from table_name (index table_name)
```

Activity on the underlying table may cause the scan to be aborted before completion.

read committed | 1 – By default, Adaptive Server's transaction isolation level is **read committed** or **1**, which allows shared read locks on data.

repeatable read | 2 – prevents nonrepeatable reads.

serializable | 3 – If you specify isolation level 3, Adaptive Server applies a **holdlock** to all **select** and **readtext** operations in a transaction, which holds the queries' read locks until the end of that transaction. If you also set chained mode, that isolation level remains in effect for any data retrieval or modification statement that implicitly begins a transaction.

transactional_rpc – controls the handling of remote procedure calls. If this option is set to **on**, when a transaction is pending, the RPC is coordinated by Adaptive Server. If this option is set to **off**, the remote procedure call is handled by the Adaptive Server site handler. The default value is inherited from the value of the **enable xact coordination** configuration parameter.

Examples

1. ```
   set showplan, noexec on
   go
   select * from publishers
   go
   ```

   For each query, returns a description of the processing plan, but does not execute it.

2. ```
   set textsize 100
   ```

   Sets the limit on *text* or *image* data returned with a **select** statement to 100 bytes.

3. ```
   set rowcount 4
   ```

   For each **insert**, **update**, **delete**, and **select** statement, Adaptive Server stops processing the query after it affects the first four rows. For example:

   ```
   select title_id, price from titles

   title_id  price
   --------  ----------
   BU1032         19.99
   BU1111         11.95
   BU2075          2.99
   BU7832         19.99

   (4 rows affected)
   ```
   ```

4. **set char_convert on with error**

   Activates character set conversion, setting it to a default based on the character set the client is using. Adaptive Server also notifies the client or application when characters cannot be converted to the client's character set.

5. **set proxy "mary"**

   The user executing this command now operates within the server as the login "mary" and Mary's server user ID.

6. **set session authorization "mary"**

   An alternative way of stating example 5.

7. **set cursor rows 5 for test_cursor**

   Returns five rows for each succeeding **fetch** statement requested by a client using *test_cursor*.

8.
```
set identity_insert stores_south on
go
insert stores_south (syb_identity)
values (100)
go
set identity_insert stores_south off
go
```

   Inserts a value of 100 into the IDENTITY column of the *stores_south* table, then prohibits further explicit inserts into this column. Note the use of the **syb_identity** keyword; Adaptive Server replaces the keyword with the name of the IDENTITY column.

9. **set transaction isolation level 3**

   Implements read-locks with each **select** statement in a transaction for the duration of that transaction.

10. **set role "sa_role" off**

    Deactivates the user's System Administrator role for the current session.

11. **set fipsflagger on**

    Tells Adaptive Server to display a warning message if you use a Transact-SQL extension. Then, if you use nonstandard SQL, like this:

```
use pubs2
go
```

    Adaptive Server displays:

```
SQL statement on line number 1 contains Non-ANSI
text. The error is caused due to the use of use
database.
```

12.set ansinull on

Tells Adaptive Server to evaluate NULL-valued operands of equality (=) and inequality (!=) comparisons and aggregate functions in compliance with the entry level SQL92 standard.

When you use **set ansinull on**, aggregate functions and row aggregates raise the following SQLSTATE warning when Adaptive Server finds null values in one or more columns or rows:

```
Warning - null value eliminated in set function
```

If the value of either the equality or the inequality operands is NULL, the comparison's result is UNKNOWN. For example, the following query returns no rows in **ansinull** mode:

```
select * from titles where price = null
```

If you use **set ansinull off**, the same query returns rows in which *price* is NULL.

13.set string_rtruncation on

Causes Adaptive Server to generate an exception when truncating a *char* or *nchar* string. If an **insert** or **update** statement would truncate a string, Adaptive Server displays:

```
string data, right truncation
```

14.set quoted_identifier on

```
go
create table "!*&strange_table"
     ("emp's_name" char(10),
     age int)
go
set quoted_identifier off
go
```

Tells Adaptive Server to treat any character string enclosed in double quotes as an identifier. The table name "!*&strange_table" and the column name "emp's_name" are legal identifier names while **quoted_identifier** is **on**.

15.set cis_rpc_handling on

Specifies that Component Integration Services handles outbound RPC requests by default.

16.set transactional_rpc on

Specifies that when a transaction is pending, the RPC is handled by the Component Integration Services access methods rather than by the Adaptive Server site handler.

17.set role doctor_role on

Activates the "doctor" role. This command is used by users to specify the roles they want activated.

18.set role doctor_role with passwd "physician" on

Activates the "doctor" role when the user enters the password.

19.set role doctor_role off

Deactivates the "doctor" role.

20.set scan_parallel_degree 4

Specifies a maximum degree of parallelism of 4 for parallel index scans and parallel table scans on nonpartitioned tables.

21.set lock wait 5

Subsequent commands in the session or stored procedure wait 5 seconds to acquire locks before generating an error message and failing.

22.set lock nowait

Subsequent commands in the session or stored procedure return an error and fail if they cannot get requested locks immediately.

23.set lock wait

Subsequent commands in the current session or stored procedure will wait indefinitely long to acquire locks.

24.set transaction isolation level 2

All subsequent queries in the session will run at the repeatable reads transaction isolation level.

25.set plan dump dev_plans on

Enables capturing abstract plans to the *dev_plans* group.

26.set plan load dev_plans on

Enables loading of abstract plans from the *dev_plans* group for queries in the current session.

27.set clientname 'alison'
set clienthostname 'money1'
set clientapplname 'webserver2'

Assigns this user:

- The client name alison
- The host name money1
- The application name webserver2

Comments

- Some **set** options can be grouped together, as follows:
  - **parseonly**, **noexec**, **prefetch**, **showplan**, **rowcount**, and **nocount** control the way a query is executed. It does not make sense to set both **parseonly** and **noexec** on. The default setting for **rowcount** is 0 (return all rows); the default for the others is **off**.
  - The **statistics** options display performance statistics after each query. The default setting for the **statistics** options is **off**. For more information about **noexec**, **prefetch**, **showplan** and **statistics**, see the *Performance and Tuning Guide*.
  - **offsets** and **procid** are used in DB-Library to interpret results from Adaptive Server. The default setting for these options is **on**.
  - **datefirst**, **dateformat**, and **language** affect date functions, date order, and message display. If used within a trigger or stored procedure, these options do not revert to their previous settings.

    In the default language, us_english, **datefirst** is 1 (Sunday), **dateformat** is *mdy*, and messages are displayed in us_english. Some language defaults (including us_english) produce Sunday=1, Monday=2, and so on; others produce Monday=1, Tuesday=2, and so on.

    set language implies that Adaptive Server should use the first weekday and date format of the language it specifies, but does not override an explicit **set datefirst** or **set dateformat** command issued earlier in the current session.
  - **cursor rows** and **close on endtran** affect the way Adaptive Server handles cursors. The default setting for **cursor rows** with all cursors is 1. The default setting for **close on endtran** is **off**.
  - **chained** and **transaction isolation level** allow Adaptive Server to handle transactions in a way that is compliant with the SQL standards.

fipsflagger, **string_rtruncation**, **ansinull**, **ansi_permissions**, **arithabort**, and **arithignore** affect aspects of Adaptive Server error handling and compliance to SQL standards.

➤ ***Note***

The **arithabort** and **arithignore** options were redefined for release 10.0 and later. If you use these options in your applications, examine them to be sure they are still producing the desired effect.

  - You can only use the **cis_rpc_handling** and **transactional_rpc** options when Component Integration Services is enabled.
  - **parallel_degree** and **scan_parallel_degree** limit the degree of parallelism for queries, if Adaptive Server is configured for parallelism. When you use these options, you give the optimizer a hint to limit parallel queries to use fewer worker processes than allowed by the configuration parameters. Setting these parameters to 0 restores the server-wide configuration values.

    If you specify a number that is greater than the numbers allowed by the configuration parameters, Adaptive Server issues a warning message and uses the value set by the configuration parameter.

- If you use the **set** command inside a trigger or stored procedure, most **set** options revert to their former settings after the trigger or procedure executes.

  The following options do not revert to their former settings after the procedure or trigger executes, but remain for the entire Adaptive Server session or until you explicitly reset them:

  - **datefirst**
  - **dateformat**
  - **identity_insert**
  - **language**
  - **quoted_identifier**

- If you specify more than one **set** option, the first syntax error causes all following options to be ignored. However, the options specified before the error are executed, and the new option values are set.
- If you assign a user a client name, host name, or application name, these assignments are only active for the current session. You will have to reasign these the next time the user logs in. Although the new names appear in *sysprocesses*, they are not used for permission checks, and **sp_who** still shows the client

connection as belonging to the original login. For more information about setting user processes, see the *System Administration Guide.*

- All **set** options except **showplan** and **char_convert** take effect immediately. **showplan** takes effect in the following batch. Here are two examples that use **set showplan on**:

```
set showplan on
select * from publishers
go
pub_id  pub_name                 city        state
------- -----------------------  ----------- -----
0736    New Age Books            Boston      MA
0877    Binnet & Hardley         Washington  DC
1389    Algodata Infosystems     Berkeley    CA

(3 rows affected)
```

But:

```
set showplan on
go
select * from publishers
go
QUERY PLAN FOR STATEMENT 1 (at line 1).
    STEP 1
        The type of query is SELECT

        FROM TABLE
            publishers
        Nested iteration
        Table Scan
        Ascending Scan.
        Positioning at start of table.

pub_id  pub_name                 city        state
------  --------------------     ----------  ----
0736    New Age Books            Boston      MA
0877    Binnet & Hardley         Washington  DC
1389    Algodata Infosystems     Berkeley    CA

(3 rows affected)
```

Roles and *set* Options

- When you log into Adaptive Server, all system-defined roles granted to you are automatically activated. User-defined roles

granted to you are not automatically activated. To automatically activate user-defined roles granted to you, use **sp_modifylogin**. For information on how to use **sp_modifylogin**, see **sp_modifylogin** in your *Adaptive Server Reference Manual*. Use **set role** *role_name* **on** or **set role** *role_name* **off** to turn roles on and off.

For example, if you have been granted the System Administrator role, you assume the identity (and user ID) of Database Owner in the current database. To assume your real user ID, execute this command:

```
set role "sa_role" off
```

If you are not a user in the current database, and if there is no "guest" user, you cannot set **sa_role off**.

- If the user-defined role you intend to activate has an attached password, you must specify the password to turn the role on. Thus, you would enter:

  ```
  set role "role_name" with passwd "password" on
  ```

Distributed Transactions, CIS, and *set* Options

- The behavior of the **cis rpc handling** configuration property and the **set transactional_rpc** commands changed with the introduction of ASTC. In previous releases, enabling **cis rpc handling** caused **all** RPCs to be routed through CIS's Client-Library connection. As a result, whenever **cis rpc handling** was enabled, **transactional_rpc** behavior occurred whether or not it had been specifically set. In Adaptive Server 12.0, this behavior has changed. If **cis rpc handling** is enabled and **transactional_rpc** is **off**, RPCs within a transaction are routed through the site handler. RPCs executed outside a transaction are sent via CIS's Client-Library connection.
- When Adaptive Server distributed transaction management services are enabled, you can place RPCs within transactions. These RPCs are called **transactional RPCs**. A transactional RPC is an RPC whose work can be included in the context of a current transaction. This remote unit of work can be committed or rolled back along with the work performed by the local transaction.

  To use transactional RPCs, enable CIS and distributed transaction management with **sp_configure**, then issue the **set transactional_rpc** command. When **set transactional_rpc** is **on** and a transaction is pending, the Adaptive Server (as opposed to the Adaptive Server site handler) coordinates the RPC.

The **set transactional_rpc** command default is **off**. The **set cis_rpc_handling** command overrides the **set transactional_rpc** command. If you set **cis_rpc_handling on**, all outbound RPCs are handled by Component Integration Services.

- See the *Component Integration Services User's Guide* for a discussion of using **set transactional_rpc**, **set cis_rpc_handling**, and **sp_configure**.

Using Proxies

- Before you can use the **set proxy** or **set session authorization** command, a System Security Officer must grant permission to execute **set proxy** or **set session authorization** from the *master* database.
- Executing **set proxy** or **set session authorization** with the original *login_name* reestablishes your previous identity.
- You cannot execute **set proxy** or **set session authorization** from within a transaction.
- Adaptive Server permits only one level of login identity change. Therefore, after you use **set proxy** or **set session authorization** to change identity, you must return to your original identity before changing it again. For example, assume that your login name is "ralph". You want to create a table as "mary", create a view as "joe", then return to your own login identity. Use the following statements:

```
set proxy "mary"
    create table mary_sales
    (stor_id  char(4),
    ord_num   varchar(20),
    date      datetime)
grant select on mary_sales to public

set proxy "ralph"

set proxy "joe"
    create view joes_view (publisher, city, state)
    as select stor_id, ord_num, date
    from mary_sales

set proxy "ralph"
```

lock wait

- By default, an Adaptive Server task that cannot immediately acquire a lock waits until incompatible locks are released, then

continues processing. This is equivalent to **set lock wait** with no value specified in the *numsecs* parameter.

- You can set a server-wide lock wait period by using the **sp_configure** system procedure with the **lock wait period** option.
- A lock wait period defined at the session level or in a stored procedure with the **set lock** command overrides a server-level lock-wait period.
- If **set lock wait** is used by itself, with no value for *numsecs*, all subsequent commands in the current session wait indefinitely to acquire requested locks.
- The **sp_sysmon** procedure reports the number of times that tasks waiting for a lock could not acquire the lock within the waiting period.

Repeatable-Reads Transaction Isolation Level

- The repeatable-reads isolation level, also known as transaction isolation level 2, holds locks on all pages read by the statement until the transaction completes.
- A nonrepeatable read occurs when one transaction reads rows from a table and a second transaction is able to modify the same rows and commit the changes before the first transaction completes. If the first transaction rereads the rows, they will have different values, so the initial read is not repeatable. Repeatable reads hold shared locks for the duration of a transaction, blocking transactions that update the locked rows or rows on the locked pages.

Using Simulated Statistics

- You can load simulated statistics into a database using the **simulate** mode of the **optdiag** utility program. If **set statistics simulate on** has been issued in a session, queries are optimized using simulated statistics, rather than the actual statistics for a table.

Global Variables Affected by *set* Options

- Table 6-34 lists the global variables that contain information about the session options controlled by the **set** command.

Table 6-34: Global variables containing session options

Global Variable	Description
@@char_convert	Contains 0 if character set conversion not in effect. Contains 1 if character set conversion is in effect.
@@isolation	Contains the current isolation level of the Transact-SQL program. *@@isolation* takes the value of the active level (0, 1 or 3).
@@options	Contains a hexadecimal representation of the session's **set** options.
@parallel_degree	Contains the current maximum parallel degree setting.
@@rowcount	Contains the number of rows affected by the last query. *@@rowcount* is set to 0 by any command that does not return rows, such as an **if** statement. With cursors, *@@rowcount* represents the cumulative number of rows returned from the cursor result set to the client, up to the last **fetch** request. *@@rowcount* is updated even when **nocount** is on.
@scan_parallel_degree	Contains the current maximum parallel degree setting for nonclustered index scans.
@@textsize	Contains the limit on the number of bytes of *text* or *image* data a **select** returns. Default limit is 32K bytes for **isql**; the default depends on the client software. Can be changed for a session with **set textsize**.
@@tranchained	Contains the current transaction mode of the Transact-SQL program. *@@tranchained* returns 0 for unchained or 1 for chained.

Using *fipsflagger* with Java in the Database

- When **fipsflagger** is on, Adaptive Server displays a warning message when these extensions are used:
  - The **installjava** utility
  - The **remove java** command
  - Column and variable declarations that reference Java classes as datatypes

- Statements that use Java-SQL expressions for member references

- The status of **fipsflagger** does not affect arithmetic expressions performed by Java methods.
- Refer to *Java in Adaptive Server Enterprise* for more information about Java in the database.

SQL92 Compliance

- The SQL92 standard specifies behavior that differs from Transact-SQL behavior in earlier Adaptive Server releases. Compliant behavior is enabled by default for all Embedded-SQL precompiler applications. Other applications needing to match this standard of behavior can use the **set** options listed in Table 6-35.

Table 6-35: Options to set for entry level SQL92 compliance

Option	Setting
ansi_permissions	on
ansinull	on
arithabort	off
arithabort numeric_truncation	on
arithignore	off
chained	on
close on endtran	on
fipsflagger	on
quoted_identifier	on
string_rtruncation	on
transaction isolation level	3

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

In general, **set** permission defaults to all users and no special permissions are required to use it. Exceptions include **set role**, **set proxy**, and **set session authorization**.

To use **set role**, a System Administrator or System Security Officer must have granted you the role. If you gain entry to a database only because you have a certain role, you cannot turn that role off while you are using the database. For example, if you are not normally authorized to use a database *info_plan*, but you use it as a System Administrator, Adaptive Server returns an error message if you try to set **sa_role off** while you are still in *info_plan*.

To use **set proxy** or **set session authorization**, you must have been granted permission by a System Security Officer.

See Also

Commands	**create trigger**, **fetch**, **insert**, **grant**, **lock table**, **revoke**
Functions	**convert**
Utilities	**isql**, **optdiag**

setuser

Function

Allows a Database Owner to impersonate another user.

Syntax

```
setuser ["user_name"]
```

Examples

```
1. setuser "mary"
   go
   grant select on authors to joe
   setuser
   go
```

The Database Owner temporarily adopts Mary's identity in the database in order to grant Joe permissions on *authors*, a table owned by Mary.

Comments

- The Database Owner uses **setuser** to adopt the identity of another user in order to use another user's database object, to grant permissions, to create an object, or for some other reason.
- When the Database Owner uses the **setuser** command, Adaptive Server checks the permissions of the user being impersonated instead of the permissions of the Database Owner. The user being impersonated must be listed in the *sysusers* table of the database.
- **setuser** affects permissions only in the local database. It does not affect remote procedure calls or accessing objects in other databases.
- The **setuser** command remains in effect until another **setuser** command is given or until the current database is changed with the **use** command.
- Executing the **setuser** command with no user name reestablishes the Database Owner's original identity.
- System Administrators can use **setuser** to create objects that will be owned by another user. However, since a System Administrator operates outside the permissions system, she or he cannot use **setuser** to acquire another user's permissions.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

setuser permission defaults to the Database Owner and is not transferable.

See Also

Commands	**grant, revoke, use**

shutdown

Function

Shuts down the Adaptive Server from which the command is issued, its local Backup Server, or a remote Backup Server. This command can be issued only by a System Administrator.

Syntax

```
shutdown [srvname] [with {wait | nowait}]
```

Keywords and Options

srvname – is the logical name by which the Backup Server is known in the Adaptive Server's *sysservers* system table. This parameter is not required when shutting down the local Adaptive Server.

with wait – is the default. This shuts down the Adaptive Server or Backup Server gracefully.

with nowait – shuts down the Adaptive Server or Backup Server immediately, without waiting for currently executing statements to finish.

➤ ***Note***

Use of **shutdown with nowai**t can lead to gaps in IDENTITY column values.

Examples

1. shutdown

Shuts down the Adaptive Server from which the **shutdown** command is issued.

2. shutdown with nowait

Shuts down the Adaptive Server immediately.

3. shutdown SYB_BACKUP

Shuts down the local Backup Server.

4. shutdown REM_BACKUP

Shuts down the remote Backup Server REM_BACKUP.

Comments

- Unless you use the **nowait** option, **shutdown** attempts to bring Adaptive Server down gracefully by:
  - Disabling logins (except for the System Administrator)
  - Performing a checkpoint in every database
  - Waiting for currently executing SQL statements or stored procedures to finish

  Shutting down the server without the **nowait** option minimizes the amount of work that must be done by the automatic recovery process.
- Unless you use the **nowait** option, **shutdown** *backup_server* waits for active dumps and/or loads to complete. Once you issue a **shutdown** command to a Backup Server, no new dumps or loads that use this Backup Server can start.
- Use **shutdown with nowait** only in extreme circumstances. In Adaptive Server, issue a **checkpoint** command before executing a **shutdown with nowait.**
- You can halt only the local Adaptive Server with **shutdown**; you cannot halt a remote Adaptive Server.
- You can halt a Backup Server only if:
  - It is listed in your *sysservers* table. The system procedure **sp_addserver** adds entries to *sysservers.*
  - It is listed in the interfaces file for the Adaptive Server where you execute the command.
- Use the **sp_helpserver** system procedure to determine the name by which a Backup Server is known to the Adaptive Server. Specify the Backup Server's *name*—not its *network_name*—as the *srvname* parameter. For example:

  sp_helpserver

```
name        network_name   status                              id
----------  -------------  ----------------------------------- --
REM_BACKUP  WHALE_BACKUP   timeouts, no net password encryption 3
SYB_BACKUP  SLUG_BACKUP    timeouts, net password encryption    1
eel         eel                                                 0
whale       whale          timeouts, no net password encryption 2
```

To shut down the remote Backup Server named WHALE_BACKUP, use the following command:

```
shutdown REM_BACKUP
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

shutdown permission defaults to System Administrators and is not transferable.

See Also

Commands	alter database
System procedures	sp_addserver, sp_helpserver

truncate table

Function

Removes all rows from a table.

Syntax

truncate table [[*database*.]*owner*.]*table_name*

Keywords and Options

table_name – is the name of the table to truncate. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

Examples

1. truncate table authors

Removes all data from the *authors* table.

Comments

- **truncate table** deletes all rows from a table. The table structure and all the indexes continue to exist until you issue a **drop table** command. The rules, defaults and constraints that are bound to the columns remain bound, and triggers remain in effect.
- **truncate table** deallocates the distribution pages for all indexes; remember to run **update statistics** after adding new rows to the table.
- **truncate table** is equivalent to—but faster than—a **delete** command without a **where** clause. **delete** removes rows one at a time and logs each deleted row as a transaction; **truncate table** deallocates whole data pages and makes fewer log entries. Both **delete** and **truncate table** reclaim the space occupied by the data and its associated indexes.
- Because the deleted rows are not logged individually, **truncate table** cannot fire a trigger.
- You cannot use **truncate table** if a another table has rows that reference it. Delete the rows from the foreign table, or truncate the foreign table, then truncate the primary table.

- You cannot use the **truncate table** command on a partitioned table. Unpartition the table with the **unpartition** clause of the **alter table** command before issuing the **truncate table** command.

  You can use the **delete** command without a **where** clause to remove all rows from a partitioned table without first unpartitioning it. This method is generally slower than **truncate table**, since it deletes one row at a time and logs each **delete** operation.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

truncate table permission defaults to the table owner and is not transferable. To truncate a system audit table (*sysaudits_01*, *sysaudits_02*, *sysaudits_03*, and so on, through *sysaudits_08*), you must be a System Security Officer.

See Also

Commands	**create trigger, delete, drop table**

union Operator

Function

Returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the **all** keyword is specified.

Syntax

```
select select_list [into clause]
          [from clause] [where clause]
          [group by clause] [having clause]
   [union [all]
       select select_list
          [from clause] [where clause]
          [group by clause] [having clause] ]...
   [order by clause]
   [compute clause]
```

Keywords and Options

union – creates the union of data specified by two **select** statements.

all – includes all rows in the results; duplicates are not removed.

into – creates a new table based on the columns specified in the select list and the rows chosen in the **where** clause. The first query in the union operation is the only one that can contain an **into** clause.

Examples

1.
```
select stor_id, stor_name from sales
union
select stor_id, stor_name from sales_east
```

   The result set includes the contents of the *stor_id* and *stor_name* columns of both the *sales* and *sales_east* tables.

2.
```
select pub_id, pub_name, city into results
from publishers
union
select stor_id, stor_name, city from stores
union
select stor_id, stor_name, city from stores_east
```

   The **into** clause in the first query specifies that the table *results* holds the final result set of the union of the specified columns of the *publishers*, *stores*, and *stores_east* tables.

3.
```
select au_lname, city, state from authors
union
((select stor_name, city, state from sales
union
select stor_name, city, state from sales_east)
union
select pub_name, city, state from publishers)
```

   First, the **union** of the specified columns in the *sales* and *sales_east* tables is generated. Then, the **union** of that result with *publishers* is generated. Finally, the **union** of the second result and *authors* is generated.

Comments

- The total number of tables that can appear on all sides of a **union** query is 256.
- The **order by** and **compute** clauses are allowed only at the end of the **union** statement to define the order of the final results or to compute summary values.
- The **group by** and **having** clauses can be used only within individual queries and cannot be used to affect the final result set.
- The default evaluation order of a SQL statement containing **union** operators is left-to-right.
- Since **union** is a binary operation, parentheses must be added to an expression involving more than two queries to specify evaluation order.
- The first query in a **union** statement may contain an **into** clause that creates a table to hold the final result set. The **into** statement must be in the first query, or you will receive an error message (see example 2).
- The **union** operator can appear within an **insert...select** statement. For example:

  ```
  insert into sales.overall
    select * from sales
    union
    select * from sales_east
  ```

- All select lists in a SQL statement must have the same number of expressions (column names, arithmetic expressions, aggregate functions, and so on). For example, the following statement is

invalid because the first select list contains more expressions than the second:

```
/* Example of invalid command--shows imbalance */
/* in select list items */
select au_id, title_id, au_ord from titleauthor
union
select stor_id, date from sales
```

- Corresponding columns in the select lists of **union** statements must occur in the same order, because **union** compares the columns one-to-one in the order given in the individual queries.
- The column names in the table resulting from a **union** are taken from the **first** individual query in the **union** statement. If you want to define a new column heading for the result set, you must do it in the first query. Also, if you want to refer to a column in the result set by a new name (for example, in an **order by** statement), you must refer to it by that name in the first **select** statement. For example, the following query is correct:

  ```
  select Cities = city from stores
  union
  select city from stores_east
  order by Cities
  ```

- The descriptions of the columns that are part of a **union** operation do not have to be identical. Table 6-36 lists the rules for the datatypes and the corresponding column in the result table.

Table 6-36: Resulting datatypes in union operations

Datatype of Columns in *union* Operation	Datatype of Corresponding Column in Result Table
Not datatype-compatible (data conversion is not handled implicitly by Adaptive Server)	Error returned by Adaptive Server.
Both are fixed-length character with lengths L1 and L2	Fixed-length character with length equal to the greater of L1 and L2.
Both are fixed-length binary with lengths L1 and L2	Fixed-length binary with length equal to the greater of L1 and L2.
Either or both are variable-length character	Variable-length character with length equal to the maximum of the lengths specified for the column in the union.

Table 6-36: Resulting datatypes in union operations (continued)

Datatype of Columns in *union* Operation	Datatype of Corresponding Column in Result Table
Either or both are variable-length binary	Variable-length binary with length equal to the maximum of the lengths specified for the columns in the union.
Both are numeric datatypes (for example, *smallint*, *int*, *float*, *money*)	A datatype equal to the maximum precision of the two columns. For example, if a column in table A is of type *int* and the corresponding column in table B is of type *float*, then the datatype of the corresponding column of the result table is *float*, because *float* is more precise than *int*.
Both column descriptions specify NOT NULL	Specifies NOT NULL.

Restrictions

- You cannot use **union** in the **select** statement of an updatable cursor.
- You cannot use the **union** operator in a **create view** statement.
- You cannot use the **union** operator in a subquery.
- You cannot use the **union** operator with the **for browse** clause.
- You cannot use the **union** operator on queries that select *text* or *image* data.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The following are Transact-SQL extensions: • The use of **union** in the select clause of an **insert** statement • Specifying new column headings in the **order by** clause of a **select** statement when the **union** operator is present in the **select** statement

See Also

Commands	**compute Clause, declare, group by and having Clauses, order by Clause, select, where Clause**
Functions	**convert**

update

Function

Changes data in existing rows, either by adding data or by modifying existing data.

Syntax

```
update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
           {expression1|NULL|(select_statement)} |
       variable_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
           {expression2|NULL|(select_statement)}]... |
       [, variable_name2 =
           {expression2|NULL|(select_statement)}]...

   [from [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}
        [,[[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}]
   ...]
   [where search_conditions]
   [plan "abstract plan"]

update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
          {expression1|NULL|(select_statement)} |
       variable_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
          {expression2|NULL|(select_statement)}]... |
       [, variable_name2 =
           {expression2|NULL|(select_statement)}]...
   where current of cursor_name
```

Keywords and Options

table_name | view_name – is the name of the table or view to update. Specify the database name if the table or view is in another

database, and specify the owner's name if more than one table or view of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

set – specifies the column name or variable name and assigns the new value. The value can be an expression or a NULL. When more than one column name or variable name and value are listed, they must be separated by commas.

from – uses data from other tables or views to modify rows in the table or view you are updating.

readpast – causes the **update** command to modify unlocked rows only on datarows-locked tables, or rows on unlocked pages, for datapages-locked tables. **update...readpast** silently skips locked rows or pages rather than waiting for the locks to be released.

where – is a standard **where** clause (see "where Clause").

index *index_name* – specifies the index to be used to access *table_name*. You cannot use this option when you update a view.

prefetch *size* – specifies the I/O size, in kilobytes, for tables bound to caches with large I/Os configured. Values for *size* are **2**, **4**, **8**, and **16**. You cannot use this option when you update a view. The procedure **sp_helpcache** shows the valid sizes for the cache to which an object is bound or for the default cache.

If Component Integration Services is enabled, you cannot use **prefetch** for remote servers.

lru | mru – specifies the buffer replacement strategy to use for the table. Use **lru** to force the optimizer to read the table into the cache on the MRU/LRU (most recently used/least recently used) chain. Use **mru** to discard the buffer from cache and replace it with the next buffer for the table. You cannot use this option when you update a view.

where current of – causes Adaptive Server to update the row of the table or view indicated by the current cursor position for *cursor_name*.

index_name – is the name of the index to be updated. If an index name is not specified, the distribution statistics for all the indexes in the specified table are updated.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan

language. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples

1.
```
update authors
set au_lname = "MacBadden"
where au_lname = "McBadden"
```

   All the McBaddens in the *authors* table are now MacBaddens.

2.
```
update titles
set total_sales = total_sales + qty
from titles, salesdetail, sales
where titles.title_id = salesdetail.title_id
    and salesdetail.stor_id = sales.stor_id
    and salesdetail.ord_num = sales.ord_num
    and sales.date in
       (select max(sales.date) from sales)
```

   Modifies the *total_sales* column to reflect the most recent sales recorded in the *sales* and *salesdetail* tables. This assumes that only one set of sales is recorded for a given title on a given date, and that updates are current.

3.
```
update titles
set price = 24.95
where current of title_crsr
```

   Changes the price of the book in the *titles* table that is currently pointed to by *title_crsr* to $24.95.

4.
```
update titles
set price = 18.95
where syb_identity = 4
```

   Finds the row for which the IDENTITY column equals 4 and changes the price of the book to $18.95. Adaptive Server replaces the **syb_identity** keyword with the name of the IDENTITY column.

5.
```
declare @x money
select @x = 0
update titles
    set total_sales = total_sales + 1,
    @x = price
    where title_id = "BU1032"
```

   Updates the *titles* table using a declared variable.

6.
```
update salesdetail set discount = 40
    from salesdetail readpast
where title_id like "BU1032"
    and qty > 100
```

Updates rows on which another task does not hold a lock.

Comments

- Use **update** to change values in rows that have already been inserted. Use **insert** to add new rows.
- You can refer to up to 15 tables in an **update** statement.
- **update** interacts with the **ignore_dup_key, ignore_dup_row**, and **allow_dup_row** options set with the **create index** command. (See **create index** for more information.)
- You can define a trigger that takes a specified action when an **update** command is issued on a specified table or on a specified column in a table.

Using Variables in *update* statements

- You can assign variables in the **set** clause of an **update** statement, similarly to setting them in a **select** statement.
- Before you use a variable in an **update** statement, you must declare the variable using **declare**, and initialize it with **select**, as shown in example 5.
- Variable assignment occurs for every qualified row in the update.
- When a variable is referenced on the right side of an assignment in an **update** statement, the current value of the variable changes as each row is updated. The **current value** is the value of the variable just before the update of the current row. The following example shows how the current value changes as each row is updated.

  Suppose you have the following statement:

```
declare @x int
select @x=0
update table1
    set C1=C1+@x, @x=@x+1
    where column2=xyz
```

The value of C1 before the update begins is 1. The following table shows how the current value of the @x variable changes after each update:

Row	Initial C1 Value	Initial @x Value	Calculations: C1+@x= updated C1	Updated C1 Value	Calculations: @x+1= updated @x	Updates Value
A	1	0	1+0	1	0+1	1
B	1	1	1+1	2	1+1	2
C	2	2	2+2	4	2+1	3
D	4	3	4+3	7	3+1	4

- When multiple variable assignments are given in the same **update** statement, the values assigned to the variables can depend on their order in the assignment list, but they might not always do so. For best results, do not rely on placement to determine the assigned values.
- If multiple rows are returned and a non-aggregating assignment of a column to a variable occurs, then the final value of the variable will be the last row process; therefore, it might not be useful.
- An **update** statement that assigns values to variables need not set the value of any qualified row.
- If no rows qualify for the update, the variable is not assigned.
- A variable that is assigned a value in the **update** statement cannot be referenced in subquery in that same **update** statement, regardless of where the subquery appears in that **update** statement.
- A variable that is assigned a value in the **update** statement cannot be referenced in a **where** or **having** clause in that same **update** statement.
- In an update driven by a join, a variable that is assigned a value in the right hand side of the **update** statement uses columns from the table that is not being updated. The result value depends on the join order chosen for the update and the number of rows that qualify from the joined table.
- Updating a variable is not affected by a rollback of the **update** statement because the value of the updated variable is not stored on disk.

Using *update* with Transactions

- When you set **chained transaction mode on**, and no transaction is currently active, Adaptive Server implicitly begins a transaction with the **update** statement. To complete the update, you must either **commit** the transaction or **rollback** the changes. For example:

```
update stores set city = 'Concord'
    where stor_id = '7066'
if exists (select t1.city, t2.city
    from stores t1, stores t2
    where t1.city = t2.city
    and t1.state = t2.state
    and t1.stor_id < t2.stor_id)
        rollback transaction
else
        commit transaction
```

  This batch begins a transaction (using chained transaction mode) and updates a row in the *stores* table. If it updates a row containing the same city and state information as another store in the table, it rolls back the changes to the *stores* table and ends the transaction. Otherwise, it commits the updates and ends the transaction.

- Adaptive Server does not prevent you from issuing an **update** statement that updates a single row more than once in a given transaction. For example, both of these updates affect the price of the book with *title_id* MC2022, since its type id "mod_cook":

```
begin transaction
update titles
set price = price + $10
where title_id = "MC2222"
update titles
set price = price * 1.1
where type = "mod_cook"
```

Using Joins in Updates

- Performing joins in the **from** clause of an **update** is an Transact-SQL extension to the ANSI standard SQL syntax for updates. Because of the way an **update** statement is processed, updates from a single statement do not accumulate. That is, if an **update** statement contains a join, and the other table in the join has more the one matching value in the join column, the second update is not based on the new values from the first update but on the original values. The results are unpredictable, since they depend on the order of processing. Consider this join:

```
update titles set total_sales = total_sales + qty
    from titles t, salesdetail sd
    where t.title_id = sd.title_id
```

The *total_sales* value is updated only once for each *title_id* in *titles*, for **one** of the matching rows in *salesdetail*. Depending on the join order for the query, on table partitioning, or on the indexes available, the results could vary each time. But each time, only a single value from *salesdetail* is added to the *total_sales* value.

If the intention is to return the sum of the values that match the join column, the following query, using a subquery, returns the correct result:

```
update titles set total_sales = total_sales +
    (select isnull(sum(qty),0)
        from salesdetail sd
        where t.title_id = sd.title_id)
    from titles t
```

Using *update* with Character Data

- Updating variable-length character data or *text* columns with the empty string ("") inserts a single space. Fixed-length character columns are padded to the defined length.
- All trailing spaces are removed from variable-length column data, except in the case of a string containing only spaces. Strings that contain only spaces are truncated to a single space. Strings longer than the specified length of a *char*, *nchar*, *varchar*, or *nvarchar* column are silently truncated unless you set **string_rtruncation on**.
- An **update** to a *text* column initializes the *text* column, assigns it a valid text pointer, and allocates at least one 2K data page.

Using *update* with Cursors

- To update a row using a cursor, define the cursor with **declare cursor**, then open it. The cursor name cannot be a Transact-SQL parameter or a local variable. The cursor must be updatable, or Adaptive Server returns an error. Any update to the cursor result set also affects the base table row from which the cursor row is derived.
- The *table_name* or *view_name* specified with an **update...where current of** must be the table or view specified in the first **from** clause of the **select** statement that defines the cursor. If that **from** clause

references more than one table or view (using a join), you can specify only the table or view being updated.

After the update, the cursor position remains unchanged. You can continue to update the row at that cursor position, provided another SQL statement does not move the position of that cursor.

- Adaptive Server allows you to update columns that are not specified in the list of columns of the cursor's *select_statement*, but that are part of the tables specified in the *select_statement*. However, when you specify a *column_name_list* with **for update,** and you are declaring the cursor, you can update only those specific columns.

Updating IDENTITY Columns

- A column with the IDENTITY property cannot be updated, either through its base table or through a view. To determine whether a column was defined with the IDENTITY property, use the **sp_help** system procedure on the column's base table.
- An IDENTITY column selected into a result table observes the following rules with regard to inheritance of the IDENTITY property:
  - If an IDENTITY column is selected more than once, it is defined as NOT NULL in the new table. It does not inherit the IDENTITY property.
  - If an IDENTITY column is selected as part of an expression, the resulting column does not inherit the IDENTITY property. It is created as NULL if any column in the expression allows nulls; otherwise, it is NOT NULL.
  - If the select statement contains a group by clause or aggregate function, the resulting column does not inherit the IDENTITY property. Columns that include an aggregate of the IDENTITY column are created NULL; others are created NOT NULL.
  - An IDENTITY column that is selected into a table with a union or join does not retain the IDENTITY property. If the table contains the union of the IDENTITY column and a NULL column, the new column is defined as NULL. Otherwise, it is defined as NOT NULL.

Updating Data Through Views

- You cannot **update** views defined with the **distinct** clause.

- If a view is created **with check option**, each row that is updated through the view must remain visible through the view. For example, the *stores_cal* view includes all rows of the *stores* table where *state* has a value of "CA". The **with check option** clause checks each **update** statement against the view's selection criteria:

  ```
  create view stores_cal
  as select * from stores
  where state = "CA"
  with check option
  ```

  An **update** statement such as this one fails if it changes *state* to a value other than "CA":

  ```
  update stores_cal
  set state = "WA"
  where store_id = "7066"
  ```

- If a view is created **with check option**, all views derived from the base view must satisfy the view's selection criteria. Each row updated through a **derived** view must remain visible through the base view.

  Consider the view *stores_cal30,* which is derived from *stores_cal.* The new view includes information about stores in California with payment terms of "Net 30":

  ```
  create view stores_cal30
  as select * from stores_cal
  where payterms = "Net 30"
  ```

  Because *stores_cal* was created **with check option**, all rows updated through *stores_cal30* must remain visible through *stores_cal*. Any row that changes *state* to a value other than "CA" is rejected.

  Notice that *stores_cal30* does not have a **with check option** clause of its own. Therefore, you can update a row with a *payterms* value other than "Net 30" through *stores_cal30*. For example, the following **update** statement would be successful, even though the row would no longer be visible through *stores_cal30*:

  ```
  update stores_cal30
  set payterms = "Net 60"
  where stor_id = "7067"
  ```

- You cannot update a row through a view that joins columns from two or more tables, unless both of the following conditions are true:

  - The view has no **with check option** clause, and
  - All columns being updated belong to the same base table.

- **update** statements are allowed on join views that contain a **with check option** clause. The update fails if any of the affected columns appear in the **where** clause in an expression that includes columns from more than one table.
- If you update a row through a join view, all affected columns must belong to the same base table.

Using *index*, *prefetch*, or *lru* | *mru*

- **index**, **prefetch**, and **lru | mru** override the choices made by the Adaptive Server optimizer. Use them with caution, and always check the performance impact with **set statistics io on**. For more information about using these options, see the *Performance and Tuning Guide*.

Using *readpast*

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.
- The **readpast** option is incompatible with the **holdlock** option. If both are specified in the same **select** command, an error is generated and the command terminates.
- If the session-wide isolation level is 3, the **readpast** option is ignored.
- If the transaction isolation level for a session is 0, **update** commands using **readpast** do not issue warning messages. For datapages-locked tables, these commands modify all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, they affect all rows that are not locked with incompatible locks.
- If an **update** command with the **readpast** option applies to two or more text columns, and the first text column checked has an incompatible lock on it, readpast locking skips the row. If the column does not have an incompatible lock, the command acquires a lock and modifies the column. Then, if any subsequent text column in the row has an incompatible lock on it, the command blocks until it can obtain a lock and modify the column.
- For more information on **readpast** locking, see the *Performance and Tuning Guide*.

Standards and Compliance

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of a **from** clause or a qualified table or column name are Transact-SQL extensions detected by the FIPS flagger. Updates through a join view or a view of which the target list contains an expression are Transact-SQL extensions that cannot be detected until run time and are not flagged by the FIPS flagger. The use of variables is a Transact-SQL extension. **readpast** is Transact-SQL extension

Permissions

update permission defaults to the table or view owner, who can transfer it to other users.

If **set ansi_permissions** is **on**, you need **update** permission on the table being updated and, in addition, you must have **select** permission on all columns appearing in the **where** clause and on all columns following the **set** clause. By default, **ansi_permissions** is **off**.

See Also

Commands	**alter table, create default, create index, create rule, create trigger, insert, where Clause**
Functions	**ptn_data_pgs**
System procedures	**sp_bindefault**, **sp_bindrule**, **sp_help**, **sp_helpartition**, **sp_helpindex**, **sp_unbindefault**, **sp_unbindrule**

update all statistics

Function

Updates all statistics information for a given table.

Syntax

```
update all statistics table_name
```

Keywords and Options

table_name – is the name of the table for which statistics are being updated.

Examples

```
1. update all statistics salesdetail
```

Updates index and partition statistics for the *salesdetail* table.

Comments

- **update all statistics** updates all statistics information for a given table. Adaptive Server keeps statistics about the distribution of pages within a table, and uses these statistics when considering whether or not to use a parallel scan in query processing on partitioned tables, and which index(es) to use in query processing. The optimization of your queries depends on the accuracy of the stored statistics.
- **update all statistics** updates statistics for all columns in a table and updates partition statistics, if the table is partitioned.
- If the table is not partitioned, **update all statistics** runs only **update statistics** on the table.
- If the table is partitioned and has no indexes, **update all statistics** runs **update partition statistics** on the table. If the table is partitioned and has indexes, **update all statistics** runs **update statistics** and **update partition statistics** on the table.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

update all statistics permission defaults to the table owner and is not transferrable.

See Also

Commands	**update statistics**, **update partition statistics**

update partition statistics

Function

Updates information about the number of pages in each partition for a partitioned table.

Syntax

```
update partition statistics table_name
   [partition_number]
```

Keywords and Options

table_name – is the name of a partitioned table.

partition_number – is the number of the partition for which you are updating information. If you do not specify a partition number, **update partition statistics** updates the number of data pages in all partitions in the specified table.

Comments

- Adaptive Server keeps statistics about the distribution of pages within a partitioned table and uses these statistics when considering whether to use a parallel scan in query processing. The optimization of your queries depends on the accuracy of the stored statistics. If Adaptive Server crashes, the distribution information could be inaccurate.

  To see if the distribution information is accurate, use the **data_pgs** function to determine the number of pages in the table, as follows:

  ```
  select data_pgs(sysindexes.id, doampg)
      from sysindexes
      where sysindexes.id = object_id("table_name")
  ```

  Then, use **sp_helpartition** on the table and add up the numbers in the "ptn_data_pgs" column of the output. The sum of the total of the number of pages that **sp_helpartition** reports should be slightly greater than the number returned by **data_pgs** because **sp_helpartition**'s page count includes OAM pages.

  If the distribution information is inaccurate, run **update partition statistics** on the table. While updating the distribution information, **update partition statistics** locks the OAM page and the control page of the partition.

- When you run **update partition statistics** on a table that contains data, or you create an index on a table that contains data, the *controlpage* column in *syspartitions* is updated to point to the control page for the partition.
- **update partition statistics** updates control page values used to estimate the number of pages in a table. These statistics are used by **sp_helpartition.**

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

update partition statistics permission defaults to the table owner and is not transferable.

See Also

Commands	alter table, update all statistics
Functions	ptn_data_pgs
System procedures	sp_helpartition

update statistics

Functionality

Updates information about the distribution of key values in specified indexes or for specified columns, for all columns in an index or for all columns in a table; allows specifying the number of steps for a histogram.

Syntax

```
update statistics table_name
    [ [index_name] | [( column_list ) ] ]
    [using step values]
    [with consumers = consumers ]

update index statistics table_name [index_name]
    [using step values]
    [with consumers = consumers ]
```

Keywords and Options

table_name – When used with **update statistics**, *table_name* is the name of the table with which the index is associated. *table_name* is required, since Transact-SQL does not require index names to be unique in a database.

index_name – is the name of the index to be updated. If an index name is not specified, the distribution statistics for all the indexes in the specified table are updated.

column_list – is a comma-separated list of columns.

using *step* **values** – specifies the number of histogram steps. The default value is 20, for columns where no statistics exist. If statistics for a column already exist in *sysstatistics*, the default value is the current number of steps.

with consumers = *consumers* – specifies the number of consumer processes to be used for a sort when *column_list* is provided and parallel query processing is enabled.

index – specifies that statistics for all columns in an index are to be updated.

Examples

1. `update statistics titles (price) using 40 values`

   Generates statistics for the *price* column of the *titles* table.

2. `update index statistics authors`

   Generates statistics for all columns in all indexes of the *authors* table.

3. `update index statistics authors au_names_ix`

   Generates statistics for all columns in the *au_names_ix* index of the *authors* table.

Comments

- Adaptive Server keeps statistics about the distribution of the key values in each index, and uses these statistics in its decisions about which index(es) to use in query processing.
- When you create a nonclustered index on a table that contains data, **update statistics** is automatically run for the new index. When you create a clustered index on a table that contains data, **update statistics** is automatically run for all indexes.
- The optimization of your queries depends on the accuracy of the statistics. If there is significant change in the key values in your index, you should rerun **update statistics** on that index or column. Use the **update statistics** command if a great deal of data in an indexed column has been added, changed, or removed (that is, if you suspect that the distribution of key values has changed).
- **update statistics**, when used with a table name and an index name, updates statistics for the leading column of an index. If **update statistics** is used with just a table name, it updates statistics for the leading columns of all indexes on the table.
- **update index statistics**, when used with a table name and an index name, updates statistics for all columns in the specified index. If **update index statistics** is used with just a table name, it updates statistics for all columns in all indexes of the table.
- Specifying the name of an unindexed column or the nonleading column of an index generates statistics for that column without creating an index.
- Specifying more than one column in a column list generates or updates a histogram for the first column, and density statistics for all prefix subsets of the list of columns.

- If you use **update statistics** to generate statistics for a column or list of columns, **update statistics** must scan the table and perform a sort.
- The **with consumers** clause is designed for use on partitioned tables on RAID devices, which appear to Adaptive Server as a single I/O device, but which are capable of producing the high throughput required for parallel sorting. For more information, see Chapter 15, "Parallel Sorting," in the *Performance and Tuning Guide*.
- Table 6-37 shows the types of scans performed during **update statistics**, the types of locks acquired, and when sorts are needed.

Table 6-37: Locking, scans, and sorts during update statistics

update statistics Specifying	Scans and Sorts Performed	Locking
Table name		
Allpages-locked table	Table scan, plus a leaf-level scan of each nonclustered index	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Table scan, plus a leaf-level scan of each nonclustered index and the clustered index, if one exists	Level 0; dirty reads
Table name and clustered index name		
Allpages-locked table	Table scan	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Leaf level index scan	Level 0; dirty reads
Table name and nonclustered index name		
Allpages-locked table	Leaf level index scan	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Leaf level index scan	Level 0; dirty reads
Table name and column name		
Allpages-locked table	Table scan; creates a worktable and sorts the worktable	Level 1; shared intent table lock, shared lock on current page
Data-only-locked table	Table scan; creates a worktable and sorts the worktable	Level 0; dirty reads

- The **update index statistics** command generates a series of update statistics operations that use the same locking, scanning, and sorting as the equivalent index-level and column-level command. For example, if the *salesdetail* table has a nonclustered index named *sales_det_ix* on *salesdetail(stor_id, ord_num, title_id)*, this command:

update index statistics salesdetail

performs these **update statistics** operations:

```
update statistics salesdetail sales_det_ix
update statistics salesdetail (ord_num)
update statistics salesdetail (title_id)
```

- The **update all statistics** commands generates a series of **update statistics** operations for each index on the table, followed by a series of **update statistics** operations for all unindexed columns, followed by an **update partition statistics** operation.
- **update statistics** is not run on the system tables in the master database during upgrade from earlier releases. Indexes exist on columns queried by most system procedures, and running **update statistics** on these tables is not required for normal usage. However, running **update statistics** is allowed on all system tables in all databases, except those that are not normal tables. These tables, which are built from internal structures when queried, include *syscurconfigs*, *sysengines*, *sysgams*, *syslisteners*, *syslocks*, *syslogs*, *syslogshold*, *sysmonitors*, *sysprocesses*, *syssecmechs*, *systestlog* and *systransactions*.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

update statistics permission defaults to the table owner and is not transferable. The command can also be executed by the Database Owner, who can impersonate the table owner by running the **setuser** command.

See Also

Commands	delete statistics

use

Function

Specifies the database with which you want to work.

Syntax

use ***database_name***

Keywords and Options

database_name – is the name of the database to open.

Examples

```
1. use pubs2
   go
```

The current database is now *pubs2*.

Comments

- The **use** command must be executed before you can reference objects in a database.
- **use** cannot be included in a stored procedure or a trigger.
- An alias permits a user to use a database under another name in order to gain access to that database. Use the system procedure **sp_addalias**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

If the database has a "guest" account, all users can use the database. If the database does not have a "guest" account, you must be a valid user in the database, have an alias in the database, or be a System Administrator or System Security Officer.

See Also

Commands	**create database, drop database**
System procedures	**sp_addalias**, **sp_adduser**, **sp_modifylogin**

waitfor

Function

Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.

Syntax

```
waitfor { delay time | time time | errorexit
   | processexit | mirrorexit }
```

Keywords and Options

delay – instructs Adaptive Server to wait until the specified amount of time has passed, up to a maximum of 24 hours.

time – instructs Adaptive Server to wait until the specified time.

time – a time in one of the acceptable formats for *datetime* data, or a variable of character type. You cannot specify dates—the date portion of the *datetime* value is not allowed.

errorexit – instructs Adaptive Server to wait until a kernel or user process terminates abnormally.

processexit – instructs Adaptive Server to wait until a kernel or user process terminates for any reason.

mirrorexit – instructs Adaptive Server to wait for a mirror failure.

Examples

```
1. begin
       waitfor time "14:20"
       insert chess(next_move)
          values('Q-KR5')
       execute sendmail 'judy'
   end
```

At 2:20 p.m., the *chess* table will be updated with my next move, and a procedure called *sendmail* will insert a row in a table owned by Judy, notifying her that a new move now exists in the *chess* table.

2.
```
declare @var char(8)
select @var = "00:00:10"
begin
    waitfor delay @var
     print "Ten seconds have passed.  Your time
       is up."
end
```

   After 10 seconds, Adaptive Server prints the message specified.

3.
```
begin
   waitfor errorexit
   print "Process exited abnormally!"
end
```

   After any process exits abnormally, Adaptive Server prints the message specified.

Comments

- After issuing the **waitfor** command, you cannot use your connection to Adaptive Server until the time or event that you specified occurs.
- You can use **waitfor errorexit** with a procedure that kills the abnormally terminated process, in order to free system resources that would otherwise be taken up by an infected process.
- To find out which process terminated, check the *sysprocesses* table with the system procedure **sp_who**.
- The time you specify with **waitfor time** or **waitfor delay** can include hours, minutes, and seconds. Use the format “hh:mi:ss”, as described in “Date and Time Datatypes” in Chapter 1, “System and User-Defined Datatypes.”

  For example:

  ```
  waitfor time "16:23"
  ```

  instructs Adaptive Server to wait until 4:23 p.m. The statement:

  ```
  waitfor delay "01:30"
  ```

  instructs Adaptive Server to wait for 1 hour and 30 minutes.
- Changes in system time (such as setting the clock back for Daylight Savings Time) can delay the **waitfor** command.
- You can use **waitfor mirrorexit** within a DB-Library program to notify users when there is a mirror failure.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

waitfor permission defaults to all users. No permission is required to use it.

See Also

Commands	**begin...end**
Datatypes	"Date and Time Datatypes"
System procedures	**sp_who**

where Clause

Function

Sets the search conditions in a **select**, **insert**, **update**, or **delete** statement.

Syntax

Search conditions immediately follow the keyword **where** in a **select**, **insert**, **update**, or **delete** statement. If you use more than one search condition in a single statement, connect the conditions with **and** or **or**.

```
where [not] expression comparison_operator expression
where [not] expression [not] like "match_string"
   [escape "escape_character"]
where [not] expression is [not] null
where [not]
   expression [not] between expression and expression
where [not]
   expression [not] in ({value_list | subquery})
where [not] exists (subquery)
where [not]
   expression comparison_operator
   {any | all} (subquery)
where [not] column_name join_operator column_name
where [not] logical_expression
where [not] expression {and | or} [not] expression
```

Keywords and Options

not – negates any logical expression or keywords such as **like**, **null**, **between**, **in**, and **exists**.

expression – is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

comparison_operator – is one of the following:

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

In comparing *char, nchar, varchar,* and *nvarchar* data, **<** means closer to the beginning of the alphabet and **>** means closer to the end of the alphabet.

Case and special character evaluations depend on the collating sequence of the operating system on the machine on which Adaptive Server is located. For example, lowercase letters may be greater than uppercase letters, and uppercase letters may be greater than numbers.

Trailing blanks are ignored for the purposes of comparison. For example, "Dirk" is the same as "Dirk ".

In comparing dates, **<** means earlier and **>** means later. Put quotes around all character and date data used with a comparison operator. For example:

```
= "Bennet"
> "94609"
```

See "User-Defined Datatypes" in Chapter 1, "System and User-Defined Datatypes," for more information about data entry rules.

like – is a keyword indicating that the following character string (enclosed by single or double quotes) is a matching pattern. **like** is available for *char, varchar, nchar, nvarchar,* and *datetime* columns, but not to search for seconds or milliseconds.

You can use the keyword **like** and wildcard characters with *datetime* data as well as with *char* and *varchar.* When you use **like** with *datetime* values, Adaptive Server converts the dates to standard *datetime* format, then to *varchar.* Since the standard storage format does not include seconds or milliseconds, you cannot search for seconds or milliseconds with **like** and a pattern.

It is a good idea to use **like** when you search for *datetime* values, since *datetime* entries may contain a variety of date parts. For example, if you insert the value "9:20" into a column named *arrival_time*, the clause:

```
where arrival_time = '9:20'
```

would not find it because Adaptive Server converts the entry into "Jan 1, 1900 9:20AM." However, the clause:

```
where arrival_time like '%9:20%'
```

would find it.

match_string – is a string of characters and wildcard characters enclosed in quotes. Table 6-38 lists the wildcard characters.

Table 6-38: Wildcard characters

Wildcard Character	Meaning
%	Any string of 0 or more characters
_	Any single character
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef])
[^]	Any single character that is not within the specified range ([^a-f]) or set ([^abcdef])

escape – specifies an escape character with which you can search for literal occurrences of wildcard characters.

escape_character – is any single character. For more information, see "Using the escape Clause" in Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

is null – searches for null values.

between – is the range-start keyword. Use **and** for the range-end value. The range:

```
where @val between x and y
```

is inclusive; the range:

```
x and @val < y
```

is not. Queries using **between** return no rows if the first value specified is greater than the second value.

and – joins two conditions and returns results when both of the conditions are true.

When more than one logical operator is used in a statement, **and** operators are usually evaluated first. However, you can change the order of execution with parentheses.

in – allows you to select values that match any one of a list of values. The comparator can be a constant or a column name, and the list can be a set of constants or, more commonly, a subquery. (See the *Transact-SQL User's Guide* for information on using **in** with a subquery.) Enclose the list of values in parentheses.

value_list – is a list of values. Put single or double quotes around character values, and separate each value from the following one with a comma (see example 7). The list can be a list of variables, for example:

```
in (@a, @b, @c)
```

However, you cannot use a variable containing a list, such as:

```
@a = "'1', '2', '3'"
```

for a values list.

exists – is used with a subquery to test for the existence of some result from the subquery. (For more information, see the *Transact-SQL User's Guide*.)

subquery – is a restricted **select** statement (**order by** and **compute** clauses and the keyword **into** are not allowed) inside the **where** or **having** clause of a **select**, **insert**, **delete**, or **update** statement, or a subquery. (For more information, see the *Transact-SQL User's Guide*.)

any – is used with **>**, **<**, or **=** and a subquery. It returns results when any value retrieved in the subquery matches the value in the **where** or **having** clause of the outer statement. (For more information, see the *Transact-SQL User's Guide*.)

all – is used with **>** or **<** and a subquery. It returns results when all values retrieved in the subquery match the value in the **where** or **having** clause of the outer statement. (For more information, see the *Transact-SQL User's Guide*.)

column_name – is the name of the column used in the comparison. Qualify the column name with its table or view name if there is

any ambiguity. For columns with the IDENTITY property, you can specify the **syb_identity** keyword, qualified by a table name where necessary, rather than the actual column name.

join_operator – is a comparison operator or one of the join operators **=*** or ***=**. (For more information, see the *Transact-SQL User's Guide*.)

logical_expression – is an expression that returns TRUE or FALSE.

or – joins two conditions and returns results when either of the conditions is true.

When more than one logical operator is used in a statement, **or** operators are normally evaluated after **and** operators. However, you can change the order of execution with parentheses.

Examples

1. `where advance * $2 > total_sales * price`
2. `where phone not like '415%'`

   Finds all the rows in which the phone number does not begin with 415.
3. `where au_lname like "[CK]ars[eo]n"`

   Finds the rows for authors named Carson, Carsen, Karsen, and Karson.
4. `where sales_east.syb_identity = 4`

   Finds the row of the *sales_east* table in which the IDENTITY column has a value of 4.
5. `where advance < $5000 or advance is null`
6. `where (type = "business" or type = "psychology")`
   `and advance > $5500`
7. `where total_sales between 4095 and 12000`
8. `where state in ('CA', 'IN', 'MD')`

   Finds the rows in which the state is one of the three in the list.

Comments

- **where** and **having** search conditions are identical, except that aggregate functions are not permitted in **where** clauses. For example, this clause is legal:

  ```
  having avg(price) > 20
  ```

  This clause is not legal:

```
where avg(price) > 20
```

See Chapter 2, "Transact-SQL Functions," for information on the use of aggregate functions, and "group by and having Clauses" for examples.

- Joins and subqueries are specified in the search conditions: see the *Transact-SQL User's Guide* for full details.
- You include up to 252 **and** and **or** conditions in a **where** clause.
- There are two ways to specify literal quotes within a *char* or *varchar* entry. The first method is to use two quotes. For example, if you began a character entry with a single quote, and you want to include a single quote as part of the entry, use two single quotes:

  ```
  'I don''t understand.'
  ```

  Or use double quotes:

  ```
  "He said, ""It's not really confusing."""
  ```

  The second method is to enclose a quote in the opposite kind of quotation mark. In other words, surround an entry containing double quotes with single quotes (or vice versa). Here are some examples:

  ```
  'George said, "There must be a better way."'
  "Isn't there a better way?"
  'George asked, "Isn"t there a better way?"'
  ```

- To enter a character string that is longer than the width of your screen, enter a backslash (\) before going to the next line.
- If a column is compared to a constant or variable in a **where** clause, Adaptive Server converts the constant or variable into the datatype of the column so that the optimizer can use the index for data retrieval. For example, *float* expressions are converted to *int* when compared to an *int* column. For example:

  ```
  where int_column = 2
  ```

  selects rows where *int_column* = 2.
- When Adaptive Server optimizes queries, it evaluates the search conditions in **where** and **having** clauses, and determines which conditions are search arguments (SARGs) that can be used to choose the best indexes and query plan. For each table in a query, a maximum of 128 search arguments can be used to optimize the query. All of the search conditions, however, are used to qualify the rows. For more information on search arguments, see the *Performance and Tuning Guide*.

- Standards and Compliance

Standard	Compliance Level
SQL92	Entry level compliant

See Also

Commands	**delete**, **execute**, **group by and having Clauses**, **insert**, **select**, **update**
Datatypes	Date and Time Datatypes
System procedures	**sp_helpjoins**

while

Function

Sets a condition for the repeated execution of a statement or statement block. The statement(s) are executed repeatedly, as long as the specified condition is true.

Syntax

```
while logical_expression [plan "abstract plan"]
      statement
```

Keywords and Options

logical_expression – is any expression that returns TRUE, FALSE, or NULL.

plan *"abstract plan"* – specifies the abstract plan to use to optimize the query. It can be a full or partial plan, specified in the abstract plan language. Plans can only be specified for optimizable SQL statements, that is, queries that access tables. See Chapter 22, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

statement – can be a single SQL statement, but is usually a block of SQL statements delimited by **begin** and **end.**

Examples

```
1. while (select avg(price) from titles) < $30
   begin
       select title_id, price
           from titles
           where price > $20
       update titles
           set price = price * 2
   end
```

If the average price is less than $30, double the prices of all books in the *titles* table. As long as it is still less than $30, the **while** loop keeps doubling the prices. In addition to determining the titles whose price exceeds $20, the **select** inside the **while** loop indicates how many loops were completed (each average result returned by Adaptive Server indicates one loop).

Comments

- The execution of statements in the **while** loop can be controlled from inside the loop with the **break** and **continue** commands.
- The **continue** command causes the **while** loop to restart, skipping any statements after the **continue**. The **break** command causes an exit from the **while** loop. Any statements that appear after the keyword **end**, which marks the end of the loop, are executed. The **break** and **continue** commands are often activated by **if** tests.

  For example:

```
while (select avg(price) from titles) < $30
begin
    update titles
        set price = price * 2
    if (select max(price) from titles) > $50
        break
    else
        if (select avg(price) from titles) > $30
            continue
    print "Average price still under $30"
end

select title_id, price from titles
    where price > $30
```

  This batch continues to double the prices of all books in the *titles* table as long as the average book price is less than $30. However, if any book price exceeds $50, the **break** command stops the **while** loop. The **continue** command prevents the **print** statement from executing if the average exceeds $30. Regardless of how the **while** loop terminates (either normally or because of the **break** command), the last query indicates which books are priced over $30.

- If two or more **while** loops are nested, the **break** command exits to the next outermost loop. All the statements after the end of the inner loop run, then the next outermost loop restarts.

◆ ***WARNING!***

If a create table or create view command occurs within a while loop, Adaptive Server creates the schema for the table or view before determining whether the condition is true. This may lead to errors if the table or view already exists.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

while permission defaults to all users. No permission is required to use it.

See Also

Commands	**begin...end, break, continue, goto Label**

writetext

Function

Permits minimally logged, interactive updating of an existing *text* or *image* column; used with the **readpast** option, skips locked rows without blocking.

Syntax

```
writetext [[database.]owner.]table_name.column_name
   text_pointer [readpast] [with log] data
```

Keywords and Options

table_name.column_name – is the name of the table and *text* or *image* column to update. Specify the database name if the table is in another database, and specify the owner's name if more than one table of that name exists in the database. The default value for *owner* is the current user, and the default value for *database* is the current database.

text_pointer – a *varbinary(16)* value that stores the pointer to the *text* or *image* data. Use the **textptr** function to determine this value, as shown in example 1. *text* and *image* data is not stored in the same set of linked pages as other table columns. It is stored in a separate set of linked pages. A pointer to the actual location is stored with the data; **textptr** returns this pointer.

readpast – specifies that the command should modify only unlocked rows. If the **writetext** command finds locked rows, it skips them, rather than waiting for the locks to be released.

with log – logs the inserted *text* or *image* data. The use of this option aids media recovery, but logging large blocks of data quickly increases the size of the transaction log, so make sure that the transaction log resides on a separate database device (see **create database**, **sp_logdevice**, and the *System Administration Guide* for details).

data – is the data to write into the *text* or *image* column. *text* data must be enclosed in quotes. *image* data must be preceded by "0x". Check the information about the client software you are using to determine the maximum length of *text* or *image* data that can be accommodated by the client.

Examples

1.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
    where au_id = "409-56-7008"
writetext blurbs.copy @val with log "hello world"
```

   This example puts the text pointer into the local variable *@val*. Then, **writetext** places the text string "hello world" into the text field pointed to by *@val*.

2.
```
declare @val varbinary(16)
select @val = textptr(copy)
from blurbs readpast
    where au_id = "409-56-7008"
writetext blurbs.copy @val readpast with log
"hello world"
```

Comments

- The maximum length of text that can be inserted interactively with **writetext** is approximately 120K bytes for *text* and *image* data.
- By default, **writetext** is a minimally logged operation; only page allocations and deallocations are logged, but the *text* or *image* data is not logged when it is written into the database. In order to use **writetext** in its default, minimally logged state, a System Administrator must use **sp_dboption** to set **select into/bulkcopy/pllsort** to **true**.
- **writetext** updates *text* data in an existing row. The update completely replaces all of the existing text.
- **writetext** operations are not caught by an **insert** or **update** trigger.
- **writetext** requires a valid text pointer to the *text* or *image* column. In order for a valid text pointer to exist, a *text* column must contain either actual data or a null value that has been explicitly entered with **update**.

  Given the table *textnull* with columns *textid* and *x*, where *x* is a *text* column that permits nulls, this **update** sets all the *text* values to NULL and assigns a valid text pointer in the *text* column:

  ```
  update textnull
  set x = null
  ```

  No text pointer results from an **insert** of an explicit null:

  ```
  insert textnull values (2,null)
  ```

And, no text pointer results from an **insert** of an implicit null:

```
insert textnull (textid)
values (2)
```

- **insert** and **update** on *text* columns are logged operations.
- You cannot use **writetext** on *text* and *image* columns in views.
- If you attempt to use **writetext** on *text* values after changing to a multibyte character set, and you have not run **dbcc fix_text**, the command fails, and an error message is generated, instructing you to run **dbcc fix_text** on the table.
- **writetext** in its default, non-logged mode runs more slowly while a **dump database** is taking place.
- The Client-Library functions **dbwritetext** and **dbmoretext** are faster and use less dynamic memory than **writetext**. These functions can insert up to 2GB of *text* data.

Using the *readpast* Option

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored.
- If the transaction isolation level for a session is 0, **writetext** commands using **readpast** do not issue warning messages. These commands at session isolation level 0 modify the specified text column if the text column is not locked with incompatible locks.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

writetext permission defaults to the table owner, who can transfer it to other users.

See Also

Commands	**readtext**
Datatypes	"text and image Datatypes"

For the Index, see volume 4, “Tables and Reference Manual Index.”

Volume 4, “Tables and Reference Manual Index,” contains the index entries for all volumes of the *Adaptive Server Reference Manual*.

For the Index, see volume 4, “Tables and Reference Manual Index.”

Sybase® Adaptive Server™ Enterprise Reference Manual

Volume 3: Procedures

Adaptive Server Enterprise Version 12
Document ID: 36273-01-1200-02
Last Revised: October 1999

Principal author: Enterprise Data Studios Publications

Contributing authors: Anneli Meyer, Evelyn Wheeler

Document ID: 36273-01-1200

This publication pertains to Adaptive Server Enterprise Version 12 of the Sybase database management software and to any subsequent version until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Sybase, the SYBASE logo, Adaptive Server, APT-FORMS, Certified SYBASE Professional, the Certified SYBASE Professional logo, Column Design, ComponentPack, Data Workbench, First Impression, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, Transact-SQL, Visual Components, VisualWriter, and VQL are registered trademarks of Sybase, Inc.

Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server Enterprise Monitor, Adaptive Warehouse, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, EnterpriseConnect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript,

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

About This Book

7. System Procedures

8. Catalog Stored Procedures

9. System Extended Stored Procedures

10. *dbcc* Stored Procedures

For the Index, see volume 4, "Tables and Reference Manual Index."

List of Figures

List of Tables

About This Book

The *Adaptive Server Reference Manual* is a four-volume guide to Sybase® Adaptive Server™ Enterprise and the Transact-SQL® language.

Volume 1, "*Building Blocks*," describes the "parts" of Transact-SQL: datatypes, built-in functions, expressions and identifiers, SQLSTATE errors, and reserved words. Before you can use Transact-SQL sucessfully, you need to understand the purpose of each of these building blocks and how its use affects the results of Transact-SQL statements.

Volume 2, "*Commands*," provides reference information about the Transact-SQL commands, which you use to create statements.

Volume 3, "*Procedures*" provides reference information about system procedures, catalog stored procedures, extended stored procedures, and **dbcc** stored procedures. All procedures are created using Transact-SQL statements.

Volume 4, "*Tables and Reference Manual Index*," provides reference information about the system tables, which store information about your server, databases, users, and other information. It also provides information about the tables in the *dbccdb* and *dbccalt* databases. It also contains an index that covers the topics of all four volumes.

For information about the intended audience of this book, related documents, other sources of information, conventions used in this manual, and help, see "About This Book" in Volume 1.

How to Use This Book

This manual contains:

- Chapter 7, "System Procedures," which provides reference pages for Adaptive Server system procedures.
- Chapter 8, "Catalog Stored Procedures," which provides reference pages for Adaptive Server catalog stored procedures.
- Chapter 9, "System Extended Stored Procedures," which provides reference pages for Adaptive Server system extended stored procedures.
- Chapter 10, "dbcc Stored Procedures," which provides reference pages for Adaptive Server **dbcc** stored procedures.

7 System Procedures

This chapter describes the system procedures, which are Sybase-supplied stored procedures used for updating and getting reports from system tables. Table 7-1 lists the system procedures discussed in this chapter.

Table 7-1: System procedures

Procedure	Description
sp_activeroles	Displays all active roles granted to a user's login.
sp_addalias	Allows an Adaptive Server user to be known in a database as another user.
sp_addauditrecord	Allows users to enter user-defined audit records (comments) into the audit trail.
sp_addaudittable	Adds another system audit table after auditing is installed.
sp_addengine	Adds an engine to an existing engine group or, if the group does not exist, creates an engine group and adds the engine.
sp_addexeclass	Creates or updates a user-defined execution class that you can bind to client applications, logins, and stored procedures.
sp_addextendedproc	Creates an extended stored procedure (ESP) in the *master* database.
sp_addexternlogin	Creates an alternate login account and password to use when communicating with a remote server through Component Integration Services.
sp_addgroup	Adds a group to a database. Groups are used as collective names in granting and revoking privileges.
sp_addlanguage	Defines the names of the months and days, and the date format, for an alternate language.
sp_addlogin	Adds a new user account to Adaptive Server.
sp_addmessage	Adds user-defined messages to *sysusermessages* for use by stored procedure **print** and **raiserror** calls and by **sp_bindmsg**.
sp_addobjectdef	Specifies the mapping between a local table and an external storage location.
sp_add_qpgroup	Adds an abstract plan group.
sp_addremotelogin	Authorizes a new remote server user by adding an entry to *master.dbo.sysremotelogins*.

Table 7-1: System procedures (continued)

Procedure	Description
sp_add_resource_limit	Creates a limit on the amount of server resources that a login or application can use to execute a query, query batch, or transaction.
sp_addsegment	Defines a segment on a database device in the current database.
sp_addserver	Defines a remote server or defines the name of the local server.
sp_addthreshold	Creates a threshold to monitor space on a database segment. When free space on the segment falls below the specified level, Adaptive Server executes the associated stored procedure.
sp_add_time_range	Adds a named time range to Adaptive Server.
sp_addtype	Creates a user-defined datatype.
sp_addumpdevice	Adds a dump device to Adaptive Server.
sp_adduser	Adds a new user to the current database.
sp_altermessage	Enables and disables the logging of a specific system-defined or user-defined message in the Adaptive Server error log.
sp_audit	Allows a System Security Officer to configure auditing options.
sp_autoconnect	Defines a passthrough connection to a remote server for a specific user, which allows the named user to enter passthrough mode automatically at login.
sp_bindcache	Binds a database, table, index, *text* object, or *image* object to a data cache.
sp_bindefault	Binds a user-defined default to a column or user-defined datatype.
sp_bindexeclass	Associates an execution class with a client application, login, or stored procedure.
sp_bindmsg	Binds a user message to a referential integrity constraint or check constraint.
sp_bindrule	Binds a rule to a column or user-defined datatype.
sp_cacheconfig	Creates, configures, reconfigures, drops, and provides information about data caches.
sp_cachestrategy	Enables or disables prefetching (large I/O) and MRU cache replacement strategy for a table, index, *text* object, or *image* object.
sp_changedbowner	Changes the owner of a database.
sp_changegroup	Changes a user's group.
sp_checknames	Checks the current database for names that contain characters not in the 7-bit ASCII set.

Table 7-1: System procedures (continued)

Procedure	Description
sp_checkreswords	Detects and displays identifiers that are Transact-SQL reserved words. Checks server names, device names, database names, segment names, user-defined datatypes, object names, column names, user names, login names, and remote login names.
sp_checksource	Checks for the existence of the **source text** of the **compiled object**.
sp_chgattribute	Changes the **max_rows_per_page** value for future space allocations of a table or index.
sp_clearpsexe	Clears the execution attributes of the client application, login, or stored procedure that was set by **sp_setpsexe**.
sp_clearstats	Initiates a new accounting period for all server users or for a specified user. Prints statistics for the previous period by executing **sp_reportstats**.
sp_cmp_all_qplans	Compares all abstract plans in two abstract plan groups.
sp_cmp_qplans	Compares two abstract plans.
sp_commonkey	Defines a common key—columns that are frequently joined—between two tables or views.
sp_companion	Performs cluster operations such as configuring Adaptive Server as a secondary companion in a high availability system and moving a companion server from one failover mode to another
sp_configure	Displays or changes configuration parameters.
sp_copy_all_qplans	Copies all plans for one abstract plan group to another group.
sp_copy_qplan	Copies one abstract plan to an abstract plan group.
sp_countmetadata	Displays the number of indexes, objects, or databases in Adaptive Server.
sp_cursorinfo	Reports information about a specific cursor or all cursors that are active for your session.
sp_dboption	Displays or changes database options.
sp_dbremap	Forces Adaptive Server to recognize changes made by **alter database**. Run this procedure only when instructed to do so by an Adaptive Server message.
sp_depends	Displays information about database object dependencies—the view(s), trigger(s), and procedure(s) that depend on a specified table or view, and the table(s) and view(s) that the specified view, trigger, or procedure depends on.
sp_deviceattr	Changes the **dsync** setting of an existing database device file.

Table 7-1: System procedures (continued)

Procedure	Description
sp_diskdefault	Specifies whether or not a database device can be used for database storage if the user does not specify a database device or specifies **default** with the **create database** or **alter database** commands.
sp_displayaudit	Displays the status of audit options.
sp_displaylevel	Sets or shows which Adaptive Server configuration parameters appear in **sp_configure** output.
sp_displaylogin	Displays information about a login account.
sp_displayroles	Displays all roles granted to another role, or displays the entire hierarchy tree of roles in table format.
sp_dropalias	Removes the alias user name identity established with **sp_addalias**.
sp_drop_all_qplans	Deletes all abstract plans in an abstract plan group.
sp_dropdevice	Drops an Adaptive Server database device or dump device.
sp_dropengine	Drops an engine from a specified engine group or, if the engine is the last one in the group, drops the engine group.
sp_dropexeclass	Drops a user-defined execution class.
sp_dropextendedproc	Removes an ESP from the master database.
sp_dropglockpromote	Removes lock promotion values from a table or database.
sp_dropgroup	Drops a group from a database.
sp_dropkey	Removes a key defined with **sp_primarykey**, **sp_foreignkey**, or **sp_commonkey** from the *syskeys* table.
sp_droplanguage	Drops an alternate language from the server and removes its row from *master.dbo.syslanguages*.
sp_droplogin	Drops an Adaptive Server user login by deleting the user's entry in *master.dbo.syslogins*.
sp_dropmessage	Drops user-defined messages from *sysusermessages*.
sp_drop_qpgroup	Drops an abstract plan group.
sp_drop_qplan	Drops an abstract plan.
sp_dropremotelogin	Drops a remote user login.
sp_drop_resource_limit	Removes one or more resource limits from Adaptive Server.
sp_dropsegment	Drops a segment from a database or unmaps a segment from a particular database device.
sp_dropserver	Drops a server from the list of known servers.

Table 7-1: System procedures (continued)

Procedure	Description
sp_dropthreshold	Removes a free-space threshold from a segment.
sp_drop_time_range	Removes a user-defined time range from Adaptive Server.
sp_droptype	Drops a user-defined datatype.
sp_dropuser	Drops a user from the current database.
sp_dumpoptimize	Specifies the amount of data dumped by Backup Server during the dump database operation.
sp_estspace	Estimates the amount of space required for a table and its indexes, and the time needed to create the index.
sp_export_qpgroup	Exports all plans for a specified user and abstract plan group to a user table.
sp_extendsegment	Extends the range of a segment to another database device.
sp_familylock	Reports information about all the locks held by a family (coordinating process and its worker processes) executing a statement in parallel.
sp_find_qplan	Finds an abstract plan, given a pattern from the query text or plan text.
sp_flushstats	Flushes statistics from in-memory storage to the *systabstats* system table.
sp_forceonline_db	Provides access to all the pages in a database that were previously taken offline by recovery.
sp_forceonline_page	Provides access to pages previously taken offline by recovery.
sp_foreignkey	Defines a foreign key on a table or view in the current database.
sp_freedll	Unloads a dynamic link library (DLL) that was previously loaded into XP Server memory to support the execution of an ESP.
sp_getmessage	Retrieves stored message strings from *sysmessages* and *sysusermessages* for **print** and **raiserror** statements.
sp_grantlogin	(Windows NT only) When Integrated Security mode or Mixed mode (with Named Pipes) is active, assigns Adaptive Server roles or **default** permissions to Windows NT users and groups.
sp_ha_admin	Performs administrative tasks on Adaptive Servers configured with Sybase Failover in a high availability system. **sp_ha_admin** is installed with the *installhavss* script (*insthasv* on Windows NT).
sp_help	Reports information about a database object (any object listed in *sysobjects*) and about Adaptive Server-supplied or user-defined datatypes.

Table 7-1: System procedures (continued)

Procedure	Description
sp_helpartition	Lists the first page and the control page for each partition in a partitioned table.
sp_helpcache	Displays information about the objects that are bound to a data cache or the amount of overhead required for a specified cache size.
sp_helpdb	Reports information about a particular database or about all databases.
sp_helpdevice	Reports information about a particular device or about all Adaptive Server database devices and dump devices.
sp_helpextendedproc	Displays ESPs registered in the current database, along with their associated DLL files.
sp_helpexternlogin	(Component Integration Services only) Reports information about external login names.
sp_helpgroup	Reports information about a particular group or about all groups in the current database.
sp_helpindex	Reports information about the indexes created on a table.
sp_helpjava	Displays information about Java classes and associated JARs that are installed in the database.
sp_helpjoins	Lists the columns in two tables or views that are likely join candidates.
sp_helpkey	Reports information about a primary, foreign, or common key of a particular table or view, or about all keys in the current database.
sp_helplanguage	Reports information about a particular alternate language or about all languages.
sp_helplog	Reports the name of the device that contains the first page of the transaction log.
sp_helpobjectdef	(Component Integration Services only) Reports information about remote object definitions. Shows owners, objects, type, and definition.
sp_help_qpgroup	Reports information on an abstract plan group.
sp_help_qplan	Reports information about an abstract plan.
sp_helpremotelogin	Reports information about a particular remote server's logins or about all remote servers' logins.
sp_help_resource_limit	Reports information about all resource limits, limits for a given login or application, limits in effect at a given time or day of the week, or limits with a given scope or action.

Table 7-1: System procedures (continued)

Procedure	Description
sp_helprotect	Reports information about permissions for database objects, users, groups, or roles.
sp_helpsegment	Reports information about a particular segment or about all segments in the current database.
sp_helpserver	Reports information about a particular remote server or about all remote servers.
sp_helpsort	Displays Adaptive Server's default sort order and character set.
sp_helptext	Prints the text of a system procedure, trigger, view, default, rule, or integrity check constraint.
sp_helpthreshold	Reports the segment, free-space value, status, and stored procedure associated with all thresholds in the current database or all thresholds for a particular segment.
sp_helpuser	Reports information about a particular user or about all users in the current database.
sp_import_qpgroup	Imports abstract plans from a user table into an abstract plan group.
sp_indsuspect	Checks user tables for indexes marked as suspect during recovery following a sort order change.
sp_listsuspect_db	Lists all databases that have offline pages because of corruption detected on recovery.
sp_listsuspect_page	Lists all pages that are currently offline because of corruption detected on recovery.
sp_lock	Reports information about processes that currently hold locks.
sp_locklogin	Locks an Adaptive Server account so that the user cannot log in, or displays a list of all locked accounts.
sp_logdevice	Moves the transaction log of a database with log and data on the same device to a separate database device.
sp_loginconfig	(Windows NT only) Displays the value of one or all integrated security parameters.
sp_logininfo	(Windows NT only) Displays all roles granted to Windows NT users and groups with **sp_grantlogin**.
sp_logiosize	Changes the log I/O size used by Adaptive Server to a different memory pool when it is doing I/O for the transaction log of the current database.
sp_modifylogin	Modifies the default database, default language, default role activation, or full name for an Adaptive Server login account.

Table 7-1: System procedures (continued)

Procedure	Description
sp_modify_resource_limit	Changes a resource limit by specifying a new limit value or the action to take when the limit is exceeded, or both.
sp_modifythreshold	Modifies a threshold by associating it with a different threshold procedure, free-space level, or segment name. You **cannot** use **sp_modifythreshold** to change the amount of free space or the segment name for the last-chance threshold.
sp_modify_time_range	Changes the start day, start time, end day, and/or end time associated with a named time range.
sp_monitor	Displays statistics about Adaptive Server.
sp_password	Adds or changes a password for an Adaptive Server login account.
sp_placeobject	Puts future space allocations for a table or an index on a particular segment.
sp_plan_dbccdb	Recommends suitable sizes for new *dbccdb* and *dbccalt* databases, lists suitable devices for *dbccdb* and *dbccalt*, and suggests a cache size and a suitable number of worker processes for the target database.
sp_poolconfig	Creates, drops, resizes, and provides information about memory pools within data caches.
sp_primarykey	Defines a primary key on a table or view.
sp_processmail	(Windows NT only) Reads, processes, sends, and deletes messages in the Adaptive Server message inbox.
sp_procqmode	Displays the query processing mode of a stored procedure, view, or trigger.
sp_procxmode	Displays or changes the transaction modes associated with stored procedures.
sp_recompile	Causes each stored procedure and trigger that uses the named table to be recompiled the next time it runs.
sp_remap	Remaps a stored procedure, trigger, rule, default, or view from releases later than 4.8 and earlier than 10.0 to be compatible with releases 10.0 and later. Use **sp_remap** on pre-release 11.0 objects that the release 11.0 upgrade procedure failed to remap.
sp_remoteoption	Displays or changes remote login options.
sp_rename	Changes the name of a user-created object or user-defined datatype in the current database.
sp_renamedb	Changes the name of a database. You **cannot** rename system databases or databases with external referential integrity constraints.

Table 7-1: System procedures (continued)

Procedure	Description
sp_rename_qpgroup	Renames an abstract plan group.
sp_reportstats	Reports statistics on system usage.
sp_revokelogin	(Windows NT only) When Integrated Security mode or Mixed mode (with Named Pipes) is active, revokes Adaptive Server roles and default permissions from Windows NT users and groups.
sp_role	Grants or revokes system roles to an Adaptive Server login account.
sp_serveroption	Displays or changes remote server options.
sp_setlangalias	Assigns or changes the alias for an alternate language.
sp_setpglockpromote	Sets or changes the lock promotion thresholds for a database, for a table, or for Adaptive Server.
sp_setpsexe	Sets custom execution attributes “on the fly” for an active client application, login, or stored procedure.
sp_set_qplan	Changes the text of the abstract plan of an existing plan without changing the associated query.
sp_setsuspect_granularity	Displays and sets the recovery fault isolation mode.
sp_setsuspect_threshold	On recovery, sets the maximum number of suspect pages that Adaptive Server will allow in the specified database before taking the entire database offline.
sp_showcontrolinfo	Displays information about engine group assignments, bound client applications, logins, and stored procedures.
sp_showexeclass	Displays the execution class attributes and the engines in any engine group associated with the specified execution class.
sp_showplan	Displays the query plan for any user connection for the current SQL statement (or a previous statement in the same batch). The query plan is displayed in **showplan** format.
sp_showpsexe	Displays execution class, current priority, and affinity for all processes running on Adaptive Server.
sp_spaceused	Displays estimates of the number of rows, the number of data pages, and the space used by one table or by all tables in the current database.
sp_syntax	Displays the syntax of Transact-SQL statements, system procedures, utilities, and other routines, depending on which products and corresponding **sp_syntax** scripts exist on Adaptive Server.
sp_sysmon	Displays performance information.

Table 7-1: System procedures (continued)

Procedure	Description
sp_thresholdaction	Executes automatically when the number of free pages on the log segment falls below the last-chance threshold, unless the threshold is associated with a different procedure. Sybase does not provide this procedure.
sp_transactions	Reports information about active transactions.
sp_unbindcache	Unbinds a database, table, index, *text* object, or *image* object from a data cache.
sp_unbindcache_all	Unbinds all objects that are bound to a cache.
sp_unbindefault	Unbinds a created default value from a column or from a user-defined datatype.
sp_unbindexeclass	Unbinds a database, table, index, *text* object, or *image* object from a data cache.
sp_unbindmsg	Unbinds a user-defined message from a constraint.
sp_unbindrule	Unbinds a rule from a column or from a user-defined datatype.
sp_volchanged	Notifies the Backup Server™ that the operator performed the requested volume handling during a dump or load.
sp_who	Reports information about all current Adaptive Server users and processes or about a particular user or process.

Introduction to System Procedures

System procedures are created by **installmaster** at installation. They are located in the *sybsystemprocs* database, and owned by the System Administrator.

Some system procedures can be run only in a specific database, but many of them can be run in any database. You can create your own system procedures that can be executed from any database. For more information, see the *System Administration Guide*.

All system procedures execute at isolation level 1.

All system procedures report a return status. For example:

```
return status = 0
```

means that the procedure executed successfully. The examples in this book do not include the return status.

Permissions on System Procedures

Permissions for system procedures are set in the *sybsystemprocs* database.

Some system procedures can be run only by Database Owners. These procedures make sure that the user executing the procedure is the owner of the database from which they are being executed.

Other system procedures (for example, all the **sp_help** procedures) can be executed by any user who has been granted permission, provided that the permission was granted in *sybsystemprocs*. A user must have permission to execute a system procedure either in all databases or in none of them.

A user who is not listed in *sybsystemprocs..sysusers* is treated as a "guest" user in *sybsystemprocs* and is automatically granted permission on many of the system procedures.

To deny a user permission on a system procedure, the System Administrator must add the user to *sybsystemprocs..sysusers* and write a **revoke** statement that applies to that procedure. The owner of a user database cannot directly control permissions on the system procedures within his or her own database.

Executing System Procedures

If a system procedure is executed in a database other than *sybsystemprocs*, it operates on the system tables in the database in which it was executed. For example, if the Database Owner of *pubs2* runs **sp_adduser** in *pubs2*, the new user is added to *pubs2..sysusers*.

To run a system procedure in a specific database, either:

- Open that database with the **use** command and execute the procedure, or
- Qualify the procedure name with the database name.

For example, the user-defined system procedure *sp_foo*, which executes the **db_name()** system function, returns the name of the database in which it is executed. When executed in the *pubs2* database, it returns the value "pubs2":

```
exec pubs2..sp_foo
------------------------------
pubs2
(1 row affected, return status = 0)
```

When executed in *sybsystemprocs*, it returns the value "sybsystemprocs":

```
exec sybsystemprocs..sp_foo

------------------------------
sybsystemprocs
(1 row affected, return status = 0)
```

Values for Parameters

If a parameter value for a system procedure contains punctuation or embedded blanks, or is a reserved word, you must enclose it in single or double quotes. If the parameter is an object name qualified by a database name or owner name, enclose the entire name in single or double quotes.

➤ ***Note***

Do not use delimited identifiers as system procedure parameters; they may produce unexpected results.

If a procedure has multiple optional parameters, you can supply parameters in the form:

```
@parametername = value
```

instead of supplying all the parameters. The parameter names in the syntax statements match the parameter names defined by the procedures.

For example, the syntax for **sp_addlogin** is:

```
sp_addlogin login_name, password [, defdb
    [, deflanguage [, fullname]]]
```

To use **sp_addlogin** to create a login for "susan" with a password of "wonderful", a full name of Susan B. Anthony, and the server's default database and language, you can use:

```
sp_addlogin susan, wonderful,
        @fullname="Susan B. Anthony"
```

This provides the same information as the command with all the parameters specified:

```
sp_addlogin susan, wonderful, public_db,
    us_english, "Susan B. Anthony"
```

You can also use "null" as a placeholder:

```
sp_addlogin susan, wonderful, null, null,
        "Susan B. Anthony"
```

Do not enclose "null" in quotes.

SQL has no rules about the number of words you can put on a line or where you must break a line. If you issue a system procedure followed by a command, Adaptive Server attempts to execute the system procedure, then the command. For example, if you execute the command:

```
sp_help checkpoint
```

Adaptive Server returns the output from **sp_help**, then runs the **checkpoint** command.

If you specify more parameters than the number of parameters expected by the system procedure, the extra parameters are ignored by Adaptive Server.

System Procedure Messages

System procedures return informational and error messages, which are listed with each procedure in this book. System procedure error messages start at error number 17000.

Error messages from the functions and commands included in a procedure are documented in *Troubleshooting and Error Messages Guide*.

System Procedure Tables

Several **system procedure tables** in the *master* database, such as *spt_values*, *spt_committab*, *spt_monitor*, and *spt_limit_types*, are used by system procedures to convert internal system values (for example, status bits) into human-readable format.

spt_values is never updated. To see how it is used, execute **sp_helptext** to look at the text for one of the system procedures that references it.

In addition, some system procedures create and then drop temporary tables.

sp_activeroles

Function

Displays all active roles.

Syntax

```
sp_activeroles [expand_down]
```

Parameters

expand_down – shows the hierarchy tree of all active roles contained by your roles.

Examples

1. **sp_activeroles**

```
Role Name
------------------------------
sa_role
sso_role
oper_role
replication_role
```

2. **sp_activeroles expand_down**

```
Role Name   Parent Role Name               Level
------------------------------ --------------------
sa_role     NULL                             1
doctor_role NULL                             1
oper_role   NULL                             1
```

Comments

- **sp_activeroles** displays all your active roles and all roles contained by those roles.
- For more information about roles, see the *System Administration Guide*.

Permissions

Any user can execute **sp_activeroles**.

Tables Used

master.dbo.sysattributes, master.dbo.syssrvroles, master.dbo.sysloginroles

See Also

Commands	**alter role**, **create role**, **drop role**, **grant**, **revoke**, **set**
Functions	**mut_excl_roles**, **proc_role**, **role_contain**, **role_name**
System procedures	**sp_displayroles**

sp_addalias

Function

Allows an Adaptive Server user to be known in a database as another user.

Syntax

```
sp_addalias loginame, name_in_db
```

Parameters

loginame – is the *master.dbo.syslogins* name of the user who wants an alternate identity in the current database.

name_in_db – is the database user name to alias *loginame* to. The name must exist in both *master.dbo.syslogins* and in the *sysusers* table of the current database.

Examples

```
1. sp_addalias victoria, albert
```

There is a user named "albert" in the database's *sysusers* table and a login for a user named "victoria" in *master.dbo.syslogins*. This command allows "victoria" to use the current database by assuming the name "albert".

Comments

- Executing **sp_addalias** maps one user to another in the current database. The mapping is shown in *sysalternates*, where the two users' *suids* (system user IDs) are connected.
- A user can be aliased to only one database user at a time.
- A report on any users mapped to a specified user can be generated with **sp_helpuser**, giving the specified user's name as an argument.
- When a user tries to use a database, Adaptive Server checks *sysusers* to confirm that the user is listed there. If the user is not listed there, Adaptive Server then checks *sysalternates*. If the user's *suid* is listed in *sysalternates*, mapped to a database user's *suid*, Adaptive Server treats the first user as the second user while using the database.

  If the user named in *loginame* is in the database's *sysusers* table, Adaptive Server does not use the user's alias identity, because it

checks *sysusers* and finds the *loginame* before checking *sysalternates*, where the alias is listed.

Permissions

Only the Database Owner or a System Administrator can execute **sp_addalias**.

Tables Used

master.dbo.syslogins, *sysalternates*, *sysobjects*, *sysusers*

See Also

Commands	**use**
System procedures	**sp_addlogin**, **sp_adduser**, **sp_dropalias**, **sp_helpuser**

sp_addauditrecord

Function

Allows users to enter user-defined audit records (comments) into the audit trail.

Syntax

```
sp_addauditrecord [text [, db_name [, obj_name
   [, owner_name [, dbid [, objid]]]]]]
```

Parameters

text – is the text of the message to add to the current audit table. The text is inserted into the *extrainfo* field of the table.

db_name – is the name of the database referred to in the record. The name is inserted into the *dbname* field of the current audit table.

obj_name – is the name of the object referred to in the record. The name is inserted into the *objname* field of the current audit table.

owner_name – is the owner of the object referred to in the record. The name is inserted into the *objowner* field of the current audit table.

dbid – is the database ID number of *db_name*. Do not enclose this integer value in quotes. *dbid* is inserted into the *dbid* field of the current audit table.

objid – is the object ID number of *obj_name*. Do not enclose this integer value in quotes. *objid* is inserted into the *objid* field of the current audit table.

Examples

1.
```
sp_addauditrecord "I gave A. Smith permission to
view the payroll table in the corporate database.
This permission was in effect from 3:10 to 3:30 pm
on 9/22/92.", "corporate", "payroll", "dbo", 10,
1004738270
```

   Adds “I gave A. Smith permission to view the payroll table in the corporate database. This permission was in effect from 3:10 to 3:30 pm on 9/22/92.” to the *extrainfo* field; “corporate” to the *dbname* field; “payroll” to the *objname* field; “dbo” to the *objowner* field; “10” to the *dbid* field, and “1004738270” to the *objid* field of the current audit table.

2. **sp_addauditrecord @text="I am disabling auditing briefly while we reconfigure the system", @db_name="corporate"**

   Adds this record to the audit trail. This example uses parameter names with the @ prefix, which allows you to leave some fields empty.

Comments

- Adaptive Server writes all audit records to the current audit table. The current audit table is determined by the value of the **current audit table** configuration parameter, set with **sp_configure**. An installation can have up to eight system audit tables, named *sysaudits_01*, *sysaudits_02*, and so forth, through *sysaudits_08*.

➤ ***Note***

The records actually are first stored in the in-memory audit queue, and the audit process later writes the records from the audit queue to the current audit table. Therefore, you cannot count on an audit record being stored immediately in the audit table.

- You can use **sp_addauditrecord** if:
  - You have been granted execute permission on **sp_addauditrecord**. (No special role is required.)
  - Auditing is enabled. (A System Security Officer used **sp_configure** to turn on the **auditing** configuration parameter.)
  - The **adhoc** option of **sp_audit** is set to **on**.

Permissions

Only a System Security Officer can execute **sp_addauditrecord**. The Database Owner of *sybsecurity* (who must also be a System Security Officer) can grant execute permission to other users.

Tables Used

sybsecurity.dbo.sysaudits_01...sysaudits_08

See Also

System procedures	**sp_audit**

sp_addaudittable

Function

Adds another system audit table after auditing is installed.

Syntax

sp_addaudittable *devname*

Parameters

devname – is the name of the device for the audit table. Specify a device name or specify "default". If you specify "default", Adaptive Server creates the audit table on the same device as the *sybsecurity* database. Otherwise, Adaptive Server creates the table on the device you specify.

Examples

1. **sp_addaudittable auditdev2**

   Creates a system audit table on *auditdev2*. If only one system audit table (*sysaudits_01*) exists when you execute the procedure, Adaptive Server names the new audit table *sysaudits_02* and places it on its own segment, called *aud_seg_02*, on *auditdev2*.

2. **sp_addaudittable "default"**

   Creates a system audit table on the same device as the *sybsecurity* database. If two system audit tables (*sysaudits_01* and *sysaudits_02*) exist when you execute the procedure, Adaptive Server names the new audit table *sysaudits_03* and places it on its own segment, called *aud_seg_03*, on the same device as the *sybsecurity* database.

Comments

- Auditing must already be installed when you run **sp_addaudittable**. Follow this procedure to add a system audit table:

  a. Create the device for the audit table, using **disk init**. For example, run a command like this for UNIX:

  ```
  disk init name = "auditdev2",
  physname = "/dev/rxyla",
  vdevno = 2, size = 5120
  ```

b. Add the device to the *sybsecurity* database with the **alter database** command. For example, to add *auditdev2* to the *sybsecurity* database, use this command:

```
alter database sybsecurity on auditdev2
```

c. Execute **sp_addaudittable** to create the table.

- Adaptive Server names the new system audit table and the new segment according to how many audit tables are already defined. For example, if five audit tables are defined before you execute the procedure, Adaptive Server names the new audit table *sysaudits_06* and the new segment *aud_seg_06*. If you specify "default", Adaptive Server places the segment on the same device as the *sybsecurity* database. Otherwise, Adaptive Server places the segment on the device you name.
- A maximum of eight audit tables is allowed. If you already have eight audit tables, and you attempt to execute **sp_addaudittable** to add another one, Adaptive Server displays an error message.
- For information about how to install auditing, see the installation documentation for your platform. For a discussion about how to use auditing, see the *System Administration Guide*.

Permissions

Only a uses who is both a System Administrator and a System Security Officer to execute **sp_addaudittable**.

Tables Used

sybsecurity..sysobjects

See Also

System procedures	**sp_audit**

sp_addengine

Function

Adds an engine to an existing engine group or, if the group does not exist, creates an engine group and adds the engine.

Syntax

sp_addengine *engine_number, engine_group*

Parameters

engine_number – is the number of the engine you are adding to the group. Legal values are between 0 and a maximum equal to the number of configured online engines minus one.

engine_group – is the name of the engine group to which you are adding the engine. If *engine_group* does not exist, Adaptive Server creates it and adds the engine to it. Engine group names must conform to the rules for identifiers. For details, see Chapter 3, "Expressions, Identifiers, and Wildcard Characters."

Examples

1. sp_addengine 2, DS_GROUP

If no engine group is called *DS_GROUP*, this statement establishes the group. If *DS_GROUP* already exists, this statement adds engine number 2 to that group.

Comments

- **sp_addengine** creates a new engine group if the value of *engine_group* does not already exist.
- The engine groups *ANYENGINE* and *LASTONLINE* are predefined. *ANYENGINE* includes all existing engines. *LASTONLINE* specifies the engine with highest engine number. A System Administrator can create additional engine groups. You cannot modify predefined engine groups.
- As soon as you use **sp_bindexeclass** to bind applications or logins to an execution class associated with *engine_group*, the associated process may start running on *engine_number*.
- Prior to making engine affinity assignments, study the environment and consider the number of non-preferred applications and the number of Adaptive Server engines

available. For more information about non-preferred applications, see the *Performance and Tuning Guide*.

Permissions

Only a System Administrator can execute **sp_addengine**.

Tables Used

master..sysattributes

See Also

System procedures	**sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**

sp_addexeclass

Function

Creates or updates a user-defined execution class that you can bind to client applications, logins, and stored procedures.

Syntax

```
sp_addexeclass classname, priority, timeslice,
    engine_group
```

Parameters

classname – is the name of the new execution class.

priority – is the priority value with which to run the client application, login, or stored procedure after it is associated with this execution class. Legal values are **HIGH**, **LOW**, and **MEDIUM**.

timeslice – is the time unit assigned to processes associated with this class. Adaptive Server currently ignores this parameter.

engine_group – identifies an existing group of engines on which processes associated with this class can run.

Examples

```
1. sp_addexeclass "DS", "LOW", 0, "DS_GROUP"
```

This statement defines a new execution class called *DS* with a *priority* value of **LOW** and associates it with the engine group *DS_GROUP*.

Comments

- **sp_addexeclass** creates or updates a user-defined execution class that you can bind to client applications, logins, and stored procedures. If the class already exists, the class attribute values are updated with the values supplied by the user.
- Use the predefined engine group parameter *ANYENGINE* if you do not want to restrict the execution object to an engine group.
- Use **sp_addengine** to define engine groups. Use **sp_showexeclass** to display execution class attributes and the engines in any engine group associated with the specified execution class. **sp_showcontrolinfo** lists the existing engine groups.

Permissions

Only a System Administrator can execute **sp_addexeclass.**

Tables Used

master..sysattributes

See Also

System procedures	**sp_addengine, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**

sp_addextendedproc

Function

Creates an extended stored procedure (ESP) in the *master* database.

Syntax

sp_addextendedproc *esp_name, dll_name*

Parameters

esp_name – is the name of the extended stored procedure. This name must be identical to the name of the procedural language function that implements the ESP. *esp_name* must be a valid Adaptive Server identifier.

dll_name – is the name of the dynamic link library (DLL) file containing the function specified by *esp_name*. The *dll_name* can be specified with no extension or with its platform-specific extension, such as *.dll* on Windows NT or *.so* on Sun Solaris. If an extension is specified, the *dll_name* must be enclosed in quotation marks.

Examples

1. `sp_addextendedproc xp_echo, "sqlsrvdll.dll"`

Registers an ESP for the function named *xp_echo*, which is in the *sqlsrvdll.dll* file. The name of the resulting ESP database object is also *xp_echo*.

Comments

- Execute **sp_addextendedproc** from the *master* database.
- The *esp_name* is case sensitive. It must match the name of the function in the DLL.
- The DLL represented by *dll_name* must reside on the server machine on which the ESP is being created and the DLL directory must be in the *$PATH* on Windows NT, the *$LD_LIBRARY_PATH* on Digital UNIX, or the *$SH_LIBRARY_PATH* on HP. If the file is not found, the search mechanism also searches *$SYBASE/dll* on Windows NT and *$SYBASE/lib* on other platforms.
- On Windows NT, an ESP function should not call a C run-time signal routine. This can cause XP Server to fail, because Open Server™ does not support signal handling on Windows NT.

Permissions

Only a System Administrator can execute **sp_addextendedproc**.

Tables Used

master.dbo.syscomments, sysobjects

See Also

Commands	**create procedure**
System procedures	**sp_dropextendedproc**, **sp_helpextendedproc**

sp_addexternlogin

(Component Integration Services only)

Function

Creates an alternate login account and password to use when communicating with a remote server through Component Integration Services.

Syntax

```
sp_addexternlogin server, loginname, externname
   [, externpassword]
```

Parameters

server – is the name of the remote server that has been added to the local server with **sp_addserver**.

loginname – is the name of the Adaptive Server login account for which to create an alternate login account.

externname – is the name of an account on the remote server *server*. This account is used when logging into *server*.

externpassword – is the password for *externname*.

Examples

1. `sp_addexternlogin JOBSERV, sa, system, sys_pass`

   Allows the local server to gain access to the remote server JOBSERV using the remote login "system" and the remote password "sys_pass" on behalf of user "sa".

2. `sp_addexternlogin CIS1012, bobj, jordan, hitchpost`

   When the user "bobj" logs into the remote server CIS1012, he connects using the remote server login name "jordan" and the password "hitchpost".

Comments

- **sp_addexternlogin** assigns an alternate login name and password to be used when communicating with a remote server. It stores the password internally in encrypted form.

  You can use **sp_addexternlogin** only when Component Integration Services is installed and configured.

- Before running **sp_addexternlogin**, add the remote server to Adaptive Server with **sp_addserver**.
- *externname* and *externpassword* must be a valid user and password combination on the node where the *server* runs.
- Sites with automatic password expiration need to plan for periodic updates of passwords for external logins.
- Use **sp_dropexternlogin** to remove the definition of the external login.
- **sp_addexternlogin** cannot be used from within a transaction.
- The "sa" account and the *loginname* account are the only users who can modify remote access for a given local user.

Permissions

Only the *loginname*, a System Administrator, and a System Security Officer can execute **sp_addexternlogin**.

Tables Used

master.dbo.syslogins, *master.dbo.sysattributes*, *master.dbo.sysservers*

See Also

System procedures	**sp_addserver**, **sp_dropexternlogin**, **sp_helpserver**

sp_addgroup

Function

Adds a group to a database. Groups are used as collective names in granting and revoking privileges.

Syntax

sp_addgroup *grpname*

Parameters

grpname – is the name of the group. Group names must conform to the rules for identifiers.

Examples

1. sp_addgroup accounting

Creates a group named *accounting* in the current database.

Comments

- **sp_addgroup** adds the new group to a database's *sysusers* table. Each group's user ID (*uid*) is 16384 or larger (except "public," which is always 0).
- A group and a user cannot have the same name.
- Once a group has been created, add new users with **sp_adduser**. To add an existing user to a group, use **sp_changegroup**.
- Every database is created with a group named "public". Every user is automatically a member of "public". Each user can be a member of one additional group.

Permissions

Only the Database Owner, a System Administrator, or a System Security Officer can execute **sp_addgroup**.

Tables Used

sysobjects, *sysusers*

See Also

Commands	**grant**, **revoke**

System procedures	**sp_adduser**, **sp_changegroup**, **sp_dropgroup**, **sp_helpgroup**

sp_addlanguage

Function

Defines the names of the months and days for an alternate language and its date format.

Syntax

```
sp_addlanguage language, alias, months, shortmons,
    days, datefmt, datefirst
```

Parameters

language – is the official language name for the language, entered in 7-bit ASCII characters only.

alias – substitutes for the alternate language's official name. Enter either "null", to make the alias the same as the official language name, or a name you prefer. You can use 8-bit ASCII characters in an alias—"français", for example—if your terminal supports them.

months – is a list of the full names of the 12 months, ordered from January through December, separated only by commas (no spaces allowed). Month names can be up to 20 characters long and can contain 8-bit ASCII characters.

shortmons – is a list of the abbreviated names of the 12 months, ordered from January through December, separated only by commas (no spaces allowed). Month abbreviations can be up to 9 characters long and can contain 8-bit ASCII characters.

days – is a list of the full names of the seven days, ordered from Monday through Sunday, separated only by commas (no spaces allowed). Day names can be up to 30 characters long and can contain 8-bit ASCII characters.

datefmt – is the date order of the date parts *month/day/year* for entering *datetime* or *smalldatetime* data. Valid arguments are *mdy*, *dmy*, *ymd*, *ydm*, *myd*, or *dym*. "dmy" indicates that dates are in day / month / year order.

datefirst – sets the number of the first weekday for date calculations. For example, Monday is 1, Tuesday is 2, and so on.

Examples

```
1. sp_addlanguage french, null,
       "janvier,fevrier,mars,avril,mai,juin,juillet,
        aout,septembre,octobre,novembre,decembre",
        "jan,fev,mars,avr,mai,juin,jui,aout,sept,oct,
        nov,dec",
        "lundi,mardi,mercredi,jeudi,vendredi,samedi,
        dimanche",
        dmy, 1
```

This stored procedure adds French to the languages available on the server. "null" makes the alias the same as the official name, "french". Date order is "dmy"—day / month / year. "1" specifies that lundi, the first item in the *days* list, is the first weekday. Because the French do not capitalize the names of the days and months except when they appear at the beginning of a sentence, this example shows them being added in lowercase.

Comments

- Usually, you add alternate languages from one of Adaptive Server's Language Modules using the **langinstall** utility or the Adaptive Server installation program. A Language Module supplies the names of the dates and translated error messages for that language. However, if a Language Module is not provided with your server, use **sp_addlanguage** to define the date names and format.
- Use **sp_modifylogin** to change a user's default language. If you set a user's default language to a language added with **sp_addlanguage**, and there are no localization files for the language, the users receive an informational message when they log in, indicating that their client software could not open the localization files.

System Table Changes

- **sp_addlanguage** creates an entry in *master.dbo.syslanguages*, inserting a unique numeric value in the *langid* column for each alternate language. *langid* 0 is reserved for U.S. English.
- The *language* parameter becomes the official language name, stored in the *name* column of *master.dbo.syslanguages*. Language names must be unique. Use **sp_helplanguage** to display a list of the alternate languages available on Adaptive Server.
- **sp_addlanguage** sets the *alias* column in *master.dbo.syslanguages* to the official language name if NULL is entered for *alias*, but System

Administrators can change the value of *syslanguage.alias* with **sp_setlangalias**.

- **sp_addlanguage** sets the *upgrade* column in *master.dbo.syslanguages* to 0.

Dates for Languages added with *sp_addlanguage*

- For alternate languages added with Language Modules, Adaptive Server sends date values to clients as *datetime* datatype, and the clients use localization files to display the dates in the user's current language. For date strings added with **sp_addlanguage**, use the **convert** function to convert the dates to character data in the server:

  ```
  select convert(char, pubdate) from table
  ```

  where *pubdate* is *datetime* data and *table* is any table.
- When users perform data entry on date values and need to use date names created with **sp_addlanguage**, the client must have these values input as character data, and sent to the server as character data.

Permissions

Only a System Administrator can execute **sp_addlanguage**.

Tables Used

master.dbo.syslanguages, *sysobjects*

See Also

Commands	**set**
System procedures	**sp_droplanguage**, **sp_helplanguage**, **sp_setlangalias**, **sp_modifylogin**

sp_addlogin

Function

Adds a new user account to Adaptive Server; specifies the password expiration interval, the minimum password length, and the maximum number of failed logins allowed for a specified login at creation.

Syntax

```
sp_addlogin loginame, passwd [, defdb]
   [, deflanguage ] [, fullname ] [, passwdexp ]
   [, minpwdlen ] [, maxfailedlogins ]
```

Parameters

loginame – is the user's login name. Login names must conform to the rules for identifiers.

passwd – is the user's password. Passwords must be at least 6 characters long. If you specify a shorter password, **sp_addlogin** returns an error message and exits. Enclose passwords that include characters besides A-Z, a-z, or 0-9 in quotation marks. Also enclose passwords that **begin** with 0-9 in quotation marks.

defdb – is the name of the default database assigned when a user logs into Adaptive Server. If you do not specify *defdb*, the default, *master*, is used.

deflanguage – is the official name of the default language assigned when a user logs into Adaptive Server. The Adaptive Server default language, defined by the **default language id** configuration parameter, is used if you do not specify *deflanguage*.

fullname – is the full name of the user who owns the login account. This can be used for documentation and identification purposes.

passwdexp – specifies the password expiration interval in days. It can be any value between 0 and 32767, inclusive.

minpwdlen – specifies the minimum password length required for that login. The values range between 0 and 30 characters.

maxfailedlogins – is the number of allowable failed login attempts. It can be any whole number between 0 and 32767.

Examples

1. **`sp_addlogin albert, longer1, corporate`**

   Creates an Adaptive Server login for "albert" with the password "longer1" and the default database *corporate*.

2. **`sp_addlogin claire, bleurouge, public_db, french`**

   Creates an Adaptive Server login for "claire". Her password is "bleurouge", her default database is *public_db*, and her default language is French.

3. **`sp_addlogin robertw, terrible2, public_db, null, "Robert Willis"`**

   Creates an Adaptive Server login for "robertw". His password is "terrible2", his default database is *public_db*, and his full name is "Robert Willis". Do not enclose **null** in quotes.

4. **`sp_addlogin susan, wonderful, null, null, "Susan B. Anthony"`**

   Creates a login for "susan" with a password of "wonderful", a full name of "Susan B. Anthony", and the server's default database and language. Do not enclose **null** in quotes.

5. **`sp_addlogin susan, wonderful, @fullname="Susan B. Anthony"`**

   An alternative way of creating the login shown in example 4.

Comments

- For ease of management, it is strongly recommended that all users' Adaptive Server login names be the same as their operating system login names. This makes it easier to correlate audit data between the operating system and Adaptive Server. Otherwise, keep a record of the correspondence between operating system and server login names.
- After assigning a default database to a user with **sp_addlogin**, the Database Owner or System Administrator must provide access to the database by executing **sp_adduser** or **sp_addalias**.
- Although a user can use **sp_modifylogin** to change his or her own default database at any time, a database cannot be used without permission from the Database Owner.
- A user can use **sp_password** at any time to change his or her own password. A System Security Officer can use **sp_password** to change any user's password.

- A user can use **sp_modifylogin** to change his or her own default language. A System Administrator can use **sp_modifylogin** to change any user's default language.
- A user can use **sp_modifylogin** to change his or her own *fullname*. A System Administrator can use **sp_modifylogin** to change any user's *fullname*.

Permissions

Only a System Administrator or a System Security Officer can execute **sp_addlogin**.

Tables Used

master.dbo.sysdatabases, master.dbo.syslogins, sysobjects

See Also

System procedures	**sp_addalias**, **sp_adduser**, **sp_droplogin**, **sp_locklogin**, **sp_modifylogin**, **sp_password**, **sp_role**

sp_addmessage

Function

Adds user-defined messages to *sysusermessages* for use by stored procedure **print** and **raiserror** calls and by **sp_bindmsg**.

Syntax

```
sp_addmessage message_num, message_text [, language
   [, with_log [, replace]]]
```

Parameters

message_num – is the message number of the message to add. The message number for a user-defined message must be 20000 or greater.

message_text – is the text of the message to add.The maximum length is 255 bytes.

language – is the language of the message to add. This must be a valid language name in the *syslanguages* table. If this parameter is missing, Adaptive Server assumes that messages are in the default session language indicated by *@@langid*.

with_log – specifies whether the message is logged in the Adaptive Server error log as well as in the Windows NT Event Log on Windows NT servers, if logging is enabled. If *with_log* is TRUE, the message is logged, regardless of the severity of the error. If *with_log* is FALSE, the message may or may not be logged, depending on the severity of the error. If you do not specify a value for *with_log*, the default is FALSE.

replace – specifies whether to overwrite an existing message of the same number and *languid*. If **replace** is specified, the existing message is overwritten; if **replace** is omitted, it is not. If you do not specify a value for **replace**, the default, FALSE, is used.

Examples

1. ```
   sp_addmessage 20001, "The table '%1!' is not owned
   by the user '%2!'."
   ```

   Adds a message with the number 20001 to *sysusermessages*.
   ```

2. **sp_addmessage 20002, "The procedure'%1!' is not owned by the user '%2!'.", NULL, TRUE, "replace"**

   Adds a message with the number 20002 to *sysusermessages*. This message is logged in the Adaptive Server error log, as well as in the Windows NT Event Log on Windows NT servers, if event logging is enabled. If a message numbered 20002 exists in the default session language, this message overwrites the old message.

Comments

- **sp_addmessage** does not overwrite an existing message of the same number and *langid* unless you specify **@replace = "replace"**.
- **print**, **raiserror**, and **sp_bindmsg** recognize placeholders in the message text to print out. A single message can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow the message when the text of the message is sent to the client.

  The placeholders are numbered to allow reordering of the arguments when Adaptive Server is translating a message to a language with a different grammatical structure. A placeholder for an argument appears as "%*nn*!", a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation point (!). The integer represents the argument number in the string in the argument list. "%1!" is the first argument in the original version, "%2!" is the second argument, and so on.

Permissions

Any user can execute **sp_addmessage**.

Tables Used

master.dbo.syslanguages, *sysobjects*, *sysusermessages*

See Also

Commands	**print**, **raiserror**
System procedures	**sp_altermessage**, **sp_dropmessage**, **sp_getmessage**

sp_addobjectdef

(Component Integration Services only)

Function

Specifies the mapping between a local table and an external storage location.

Syntax

```
sp_addobjectdef tablename, "objectdef" ["objecttype"]
```

Parameters

tablename – is the name of the object as it is defined in a local table. The *tablename* can be in any of the following forms:

- *dbname.owner.object*
- *dbname..object*
- *owner.object*
- *object*

dbname and *owner* are optional. *object* is required. If you do not specify an *owner*, the default (current user name) is used. If you specify a *dbname*, it must be the current database name, and you must specify *owner* or mark the owner with a placeholder in the format *dbname..object*. Enclose any multipart *tablename* values in quotes.

objectdef – is a string naming the external storage location of the object. The *objecttype* at *objectdef* can be a table, view, or read-only remote procedure call (RPC) result set accessible to a remote server. A table, view, or RPC uses the following format for *objectdef*:

server_name.dbname.owner.object

server_name and *object* are required. *dbname* and *owner* are optional, but if they are not supplied, a placeholder in the format *dbname..object*, is required.

For more information, see “Server Classes” in the *Component Integration Services User’s Guide*.

objecttype – is one of the values that specify the format of the object named by *objectdef*. Table 7-2 describes the valid values. Enclose the *objecttype* value in quotes.

Table 7-2: Allowable values for objecttype

Value	Description
table	Indicates that the object named by *objectdef* is a table accessible to a remote server. This value is the default for *objecttype*.
view	Indicates that the object named by *objectdef* is a view managed by a remote server and processed as a table.
rpc	Indicates that the object named by *objectdef* is an RPC managed by a remote server. Adaptive Server processes the result set from the RPC as a read-only table.

Table 7-3 summarizes how each *objecttype* is used:

Table 7-3: Summary of objecttype uses

objecttype	*create table*	*create existing table*	Write to Table	Read from Table
table	Yes	Yes	Yes	Yes
view	No	Yes	Yes	Yes
rpc	No	Yes	No	Yes

Examples

1. **`sp_addobjectdef "finance.dbo.accounts", "SYBASE.pubs.dbo.accounts", "table"`**

   Maps the local table *accounts* in the database *finance* to the remote object *pubs.dbo.accounts* in the remote server named SYBASE. The current database must be *finance*. A subsequent **create table** creates a table in the *pubs* database. If *pubs.dbo.accounts* is an existing table, a **create existing table** statement populates the table *finance.dbo.accounts* with information about the remote table.

2. **`sp_addobjectdef stockcheck, "NEWYORK.wallstreet.kelly.stockcheck", "rpc"`**

   Maps the local table *stockcheck* to an RPC named *stockcheck* on remote server NEWYORK in the database *wallstreet* with owner

"kelly". The result set from RPC *stockcheck* is seen as a read-only table. Typically, the next operation would be a **create existing table** statement for the object *stockcheck*.

Comments

- **sp_addobjectdef** specifies the mapping between a local table and an external storage location. It identifies the format of the object at that location.

  You can use **sp_addobjectdef** only when Component Integration Services is installed and configured.

- **sp_addobjectdef** replaces the **sp_addtabledef** command. **sp_addtabledef** allows existing scripts to run without modification. Internally, **sp_addtabledef** invokes **sp_addobjectdef.**

- Only the System Administrator can provide the name of another user as a table owner.

- When *objecttype* is **table**, **view**, or **rpc**, the *objectdef* parameter takes the following form:

  "*server_name.database.owner.tablename*"

  - *server_name* represents a server that has already been added to *sysservers* by **sp_addserver.**
  - *database* may not be required. Some server classes do not support it.
  - *owner* should always be provided, to avoid ambiguity. If you do not specify *owner*, the remote object referenced may vary, depending on whether or not the external login corresponds to the remote object owner.
  - *tablename* is the name of a remote server table.

- Use **sp_addobjectdef** before issuing any **create table** or **create existing table** commands. **create table** is valid only for the *objecttype* values **table** and **file**. When either **create table** or **create existing table** is used, Adaptive Server checks *sysattributes* to determine whether any table mapping has been specified for the object. Follow the *objecttype* values **view** and **rpc** with **create existing table** statements.

- After the table has been created, all future references to the local table name (by **select, insert, delete** and **update**) are mapped to the correct location.

- For information about RMS, see the *Component Integration Services User's Guide.*

Permissions

Any user can execute **sp_addobjectdef**.

Tables Used

sysobjects, *sysattributes*, *sysservers*

See Also

Commands	**create existing table**, **create table**, **drop table**
System procedures	**sp_addlogin**, **sp_addserver**, **sp_defaultloc**, **sp_dropobjectdef**, **sp_helpserver**

sp_add_qpgroup

Function

Adds an abstract plan group.

Syntax

```
sp_add_qpgroup new_name
```

Parameters

new_name – is the name of the new abstract plan group. Group names must be valid identifiers.

Examples

```
1. sp_add_qpgroup dev_plans
```

Creates a new abstract plan group named *dev_plans*.

Comments

- Use **sp_add_qpgroup** to add abstract plan groups for use in capturing or creating abstract plans. The abstract plan group must exist before you can create, save, or copy plans into a group.
- **sp_add_qpgroup** cannot be run in a transaction.

Permissions

Only a System Administrator or Database Owner can execute **sp_add_qpgroup**.

Tables Used

sysattributes

See Also

Commands	**set plan**
System procedures	**sp_help_qpgroup**

sp_addremotelogin

Function

Authorizes a new remote server user by adding an entry to *master.dbo.sysremotelogins*.

Syntax

```
sp_addremotelogin remoteserver [, loginame
   [, remotename] ]
```

Parameters

remoteserver – is the name of the remote server to which the remote login applies. This server must be known to the local server by an entry in the *master.dbo.sysservers* table, which was created with **sp_addserver**.

➤ ***Note***

This manual page uses the term "local server" to refer to the server that is executing the remote procedures run from a "remote server".

loginame – is the login name of the user on the local server. *loginame* must already exist in the *master.dbo.syslogins* table.

remotename – is the name used by the remote server when logging into the local server. All *remotename*s that are not explicitly matched to a local *loginame* are automatically matched to a local name. In example 1, the local name is the remote name that is used to log in. In example 2, the local name is “albert”.

Examples

1. `sp_addremotelogin GATEWAY`

Creates an entry in the *sysremotelogins* table for the remote server GATEWAY, for purposes of login validation. This is a simple way to map remote names to local names when the local and remote servers have the same users.

This example results in a value of -1 for the *suid* column and a value of NULL for the *remoteusername* in a row of *sysremotelogins*.

2. **`sp_addremotelogin GATEWAY, albert`**

   Creates an entry that maps all logins from the remote server GATEWAY to the local user name "albert". Adaptive Server adds a row to *sysremotelogins* with Albert's server user ID in the *suid* column and a null value for the *remoteusername*.

   For these logins to be able to run RPCs on the local server, they must specify a password for the RPC connection when they log into the local server, or they must be "trusted" on the local server. To define these logins as "trusted", use **sp_remoteoption**.

3. **`sp_addremotelogin GATEWAY, ralph, pogo`**

   Maps a remote login from the remote user "pogo" on the remote server GATEWAY to the local user "ralph". Adaptive Server adds a row to *sysremotelogins* with Ralph's server user ID in the *suid* column and "pogo" in the *remoteusername* column.

Comments

- When a remote login is received, the local server tries to map the remote user to a local user in three different ways:
  - First, the local server looks for a row in *sysremotelogins* that matches the remote server name and the remote user name. If the local server finds a matching row, the local server user ID for that row is used to log in the remote user. This applies to mappings from a specified remote user.
  - If no matching row is found, the local server searches for a row that has a null remote name and a local server user ID other than -1. If such a row is found, the remote user is mapped to the local server user ID in that row. This applies to mappings from any remote user from the remote server to a specific local name.
  - Finally, if the previous attempts failed, the local server checks the *sysremotelogins* table for an entry that has a null remote name and a local server user ID of -1. If such a row is found, the local server uses the remote name supplied by the remote server to look for a local server user ID in the *syslogins* table. This applies when login names from the remote server and the local server are the same.
- The name of the local user may be different on the remote server.
- If you use **sp_addremotelogin** to map all users from a remote server to the same local name, use **sp_remoteoption** to specify the "trusted" option for those users. For example, if all users from the server

GOODSRV that are mapped to "albert" are to be "trusted", use **sp_remoteoption** as follows:

```
sp_remoteoption GOODSRV, albert, NULL, trusted,
  true
```

Logins that are not specified as "trusted" cannot execute RPCs on the local server unless they specify passwords for the local server when they log into the remote server. In Open Client™ Client-Library™, the user can use the **ct_remote_pwd** routine to specify a password for server-to-server connections. **isql** and **bcp** do not permit users to specify a password for RPC connections.

If users are logged into the remote server using "unified login", these logins are already authenticated by a security mechanism. These logins must also be trusted on the local server, or the users must specify passwords for the server when they log into the remote server.

- For more information about setting up servers for remote procedure calls and for using "unified login", see the *System Administration Guide*.
- Every remote login entry has a status. The default status for the **trusted** option is **false** (not trusted). This means that when a remote login comes in using that entry, the password is checked. If you do not want the password to be checked, change the status of the **trusted** option to **true** with **sp_remoteoption**.

Permissions

Only a System Administrator can execute **sp_addremotelogin**.

Tables Used

master.dbo.syslogins, *master.dbo.sysremotelogins*, *master.dbo.sysservers*, *sysobjects*

See Also

System procedures	**sp_addlogin**, **sp_addserver**, **sp_dropremotelogin**, **sp_helpremotelogin**, **sp_helprotect**, **sp_helpserver**, **sp_remoteoption**
Utility	**isql**

sp_add_resource_limit

Function

Creates a limit on the number of server resources that can be used by an Adaptive Server login and / or an application to execute a query, query batch, or transaction.

Syntax

```
sp_add_resource_limit name, appname, rangename,
   limittype, limitvalue [, enforced [, action
   [, scope ]]]
```

Parameters

name – is the Adaptive Server login to which the limit applies. You must specify either a *name* or an *appname* or both. To create a limit that applies to all users of a particular application, specify a *name* of NULL.

appname – is the name of the application to which the limit applies. You must specify either a *name* or an *appname* or both. To create a limit that applies to all applications used by an Adaptive Server login, specify an *appname* of **null**. To create a limit that applies to a particular application, specify the application name that the client program passes to the Adaptive Server in the login packet.

rangename – is the time range during which the limit is enforced. The time range must exist in the *systimeranges* system table of the *master* database at the time you create the limit.

limittype – is the type of resource to limit. This must be one of the following:

Limit Type	Description
row_count	Limits the number of rows a query can return
elapsed_time	Limits the number of seconds, in wall-clock time, that a query batch or transaction can run
io_cost	Limits either the actual cost or the optimizer's cost estimate for processing a query

limitvalue – is the maximum amount of the server resource (I / O cost, elapsed time in seconds, or row count) that can be used by the

login or application before Adaptive Server enforces the limit. This must be a positive, nonzero integer that is less than or equal to 2^{31}. The following table indicates what value to specify for each limit type:

Limit Type	Limit Value
row_count	The maximum number of rows that can be returned by a query before the limit is enforced.
elapsed_time	The number of seconds, in wall-clock time, that a query batch or transaction can run before the limit is enforced.
io_cost	A unitless measure derived from the optimizer's costing formula.

enforced – determines whether the limit is enforced prior to or during query execution. The following table lists the valid values for each limit type:

enforced Code	Description	Limit Type
1	Action is taken when the estimated I/O cost of execution exceeds the specified limit.	**io_cost**
2	Action is taken when the actual row count, elapsed time, or I/O cost of execution exceeds the specified limit.	**row_count** **elapsed_time** **io_cost**
3	Action is taken when either the estimated cost or the actual cost exceeds the specified limit.	**io_cost**

If you specify an *enforced* value of 3, Adaptive Server performs a logical "or" of 1 and 2. For example, assume *enforced* is set to 3. If you run a query whose **io_cost** exceeds the estimated cost, the specified *action* is executed. If the query is within the limits specified for estimated cost but exceeds the actual cost, the specified *action* is also executed.

If you do not specify an *enforced* value, Adaptive Server enforces limit 2 for **row_count** and **elapsed_time** and limit 3 for **io_cost.** In other words, if the limit type is **io_cost**, the specified action is executed if the query exceeds either the estimated or actual cost.

action – is the action to take when the limit is exceeded. The following action codes are valid for all limit types:

action Code	Description
1	Issues a warning
2	Aborts the query batch
3	Aborts the transaction
4	Kills the session

If you do not specify an *action* value, Adaptive Server uses a default value of 2 (abort the query batch).

scope – is the scope of the limit. Specify one of the following codes appropriate to the type of limit:

scope Code	Description	Limit Type
1	Query	**io_cost** **row_count**
2	Query batch (one or more SQL statements sent by the client to the server)	**elapsed_time**
4	Transaction	**elapsed_time**
6	Query batch **and** transaction	**elapsed_time**

If you do not specify a *scope* value, the limit applies to all possible scopes for the limit type.

Examples

1. **`sp_add_resource_limit NULL, payroll, early_morning, elapsed_time, 120, 2, 1, 2`**

   Creates a resource limit that applies to all users of the **payroll** application during the *early_morning* time range. If the query batch takes more than 120 seconds to execute, Adaptive Server issues a warning.

2. **`sp_add_resource_limit joe_user, NULL, midday, row_count, 5000, 2, 3, 1`**

   Creates a resource limit that applies to all ad hoc queries and applications run by “joe_user” during the *midday* time range.

When a query returns more than 5000 rows, Adaptive Server aborts the transaction.

3. **`sp_add_resource_limit joe_user, NULL, midday, io_cost, 650, 1, 3, 1`**

   Creates a resource limit that applies to all ad hoc queries and applications run by "joe_user" during the *midday* time range. When the optimizer estimates that the I/O cost would exceed 650, Adaptive Server aborts the transaction.

Comments

- You must enable **sp_configure "allow resource limits"** for resource limits to take effect.
- Multiple resource limits can exist for a given user, application, limit type, scope, and enforcement time, as long as their time ranges do not overlap.
- All limits for the currently active named time ranges and the "at all times" range for a login and/or application name are bound to the user's session at login time. Therefore, if a user logs into Adaptive Server independently of a given application, resource limits that restrict the user in combination with that application do not apply. To guarantee restrictions on that user, create a resource limit that is specific to the user and independent of any application.
- Since either the user login name or application name, or both, are used to identify a resource limit, Adaptive Server observes a predefined search precedence while scanning the *sysresourcelimits* table for applicable limits for a login session. The following table describes the precedence of matching ordered pairs of login name and application name:

Level	Login Name	Application Name
1	"joe_user"	**payroll**
2	NULL	**payroll**
3	"joe_user"	NULL

If one or more matches are found for a given precedence level, no further levels are searched. This prevents conflicts regarding similar limits for different login/application combinations.

If no match is found at any level, no limit is imposed on the session.

- When you add, delete, or modify resource limits, Adaptive Server rebinds the limits for each session for that login and/or application at the beginning of the next query batch for that session.
- When you change the currently active time ranges, Adaptive Server rebinds limits for the session. This rebinding occurs at the beginning of the next query batch.
- You cannot associate the limits for a particular login, application, or login/application combination with named time ranges that overlap (except for limits that share the same time range).

  For example, if a user is limited to retrieving 50 rows between 9:00 a.m. and 1:00 p.m., you cannot create a second resource limit for the same user that limits him to retrieving 100 rows between 10:00 a.m. and 12:00 noon. However, you can create a resource hierarchy by assigning the 100-row limit to the **user** between 10:00 a.m. and 12:00 noon and assigning the 50-row limit to an **application**, like **isql**, between 9:00 a.m. and 1:00 p.m.
- For more information on resource limits, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_add_resource_limit**.

Tables Used

master..sysresourcelimits, master..systimeranges

See Also

System procedures	**sp_configure**, **sp_drop_resource_limit**, **sp_help_resource_limit**, **sp_modify_resource_limit**
Utility	**isql**

sp_addsegment

Function

Defines a segment on a database device in a database.

Syntax

sp_addsegment *segname*, *dbname*, *devname*

Parameters

segname – is the name of the new segment to add to the *syssegments* table of the database. Segment names are unique in each database.

dbname – specifies the name of the database in which to define the segment. *dbname* must be the name of the current database or match the database name qualifying **sp_addsegment**.

devname – is the name of the database device in which to locate *segname*. A database device can have more than one segment associated with it.

Examples

1. sp_addsegment indexes, pubs2, dev1

Creates a segment named *indexes* for the database *pubs2* on the database device named *dev1*.

```
2. disk init
        name = "pubs2_dev",
        physname = "/dev/pubs_2_dev",
        vdevno = 9, size = 5120
   go
   alter database pubs2 on pubs2_dev = 2
   go
   pubs2..sp_addsegment indexes, pubs2, dev1
```

Creates a segment named *indexes* for the database *pubs2* on the database device named *dev1*.

Comments

- **sp_addsegment** defines segment names for database devices created with **disk init** and assigned to a specific database with an **alter database** or **create database** command.

- After defining a segment, use it in **create table** and **create index** commands and in the **sp_placeobject** procedure to place a table or index on the segment.

  When a table or index is created on a particular segment, all subsequent data for the table or index is located on the segment.
- Use the system procedure **sp_extendsegment** to extend the range of a segment to another database device used by the same database.
- If a database is extended with **alter database** on a device used by that database, the segments mapped to that device are also extended.
- The *system* and *default* segments are mapped to each database device included in a **create database** or **alter database** command. The *logsegment* is also mapped to each device, unless you place it on a separate device with the **log on** extension to **create database** or with **sp_logdevice**. For more information, see the *System Administration Guide*.
- If you attempt to use **sp_addsegment** in a database that has both data and the log on the same device, Adaptive Server returns an error message.

Permissions

Only the Database Owner or a System Administrator can execute **sp_addsegment**.

Tables Used

master.dbo.sysdevices, *master.dbo.sysusages*, *sysobjects*, *syssegments*

See Also

Commands	**alter database**, **create index**, **create table**, **disk init**
System procedures	**sp_dropsegment**, **sp_extendsegment**, **sp_helpdb**, **sp_helpdevice**, **sp_placeobject**

sp_addserver

Function

Defines a remote server, or defines the name of the local server.

Syntax

```
sp_addserver lname [, class [, pname]]
```

Parameters

lname – is the name used to address the server on your system. **sp_addserver** adds a row to the *sysservers* table if there is no entry already present for *lname*. Server names must be unique and must conform to the rules for identifiers.

class – identifies the category of server being added. Table 7-4 lists allowable values for the *class* parameter:

Table 7-4: Allowable values for server_class parameter

class Parameter Value	Description
access_server	Server coded to the DirectConnect™ specification (Component Integration Services only)
db2	Server accessible by Net-Gateway™ or MDI™ Database Gateway (Component Integration Services only)
direct_connect	Functionally the same as **access_server** (Component Integration Services only)
generic	Server coded to the Generic Access Module specification (Component Integration Services only)
local	Local server (there can be only one) used only once after start-up, or after restarting Adaptive Server, to identify the local server name so that it can appear in messages printed by Adaptive Server
null	Remote server with no category defined
sql_server	Another Adaptive Server or Omni server (this is the default value)

pname – is the name in the interfaces file for the server named *lname*. This enables you to establish local aliases for other Adaptive

Servers or Backup Servers that you may need to communicate with. If you do not specify a *pname*, *lname* is used.

Examples

1. **`sp_addserver GATEWAY`**

   Adds an entry for a remote server named GATEWAY in *master.dbo.sysservers*. The *pname* is also GATEWAY.

2. **`sp_addserver GATEWAY, null, VIOLET`**

   Adds an entry for a remote server named GATEWAY in *master.dbo.sysservers*. The *pname* is VIOLET. If there is already a *sysservers* entry for GATEWAY with a different *pname*, the *pname* of server GATEWAY changes to VIOLET.

3. **`sp_addserver PRODUCTION, local`**

   Adds an entry for the local server named PRODUCTION.

4. **`sp_addserver SQLSRV10, sql_server, SS_MOSS`**

   Adds an entry for a remote server known to the local server as SQLSRV10. The remote server is of server class **sql_server**. The *network_name* for SQLSRV10 is SS_MOSS.

5. **`sp_addserver RDBAM_ALPHA, generic, rdbam_alpha`**

   Adds an entry for a remote server known to the local server as *RDBAM_ALPHA*. The remote server *RDBAM_ALPHA* is written to the Generic Access Module specification, which requires server class **generic**.

Comments

- The *sysservers* table identifies the name of the local server and its options, and any remote servers that the local server can communicate with.

  To execute a remote procedure call on a remote server, the remote server must exist in the *sysservers* table.

- If *lname* already exists as a server name in the *sysservers* table, **sp_addserver** changes the remote server's *srvnetname* to the name specified by *pname*. When it does this, **sp_addserver** reports which server it changed, what the old network name was, and what the new network name is.

- The installation or upgrade process for your server adds an entry in *sysservers* for a Backup Server. If you remove this entry, you cannot back up your databases.

- Adaptive Server requires that the Backup Server have an *lname* of SYB_BACKUP. If you do not want to use that as the name of your Backup Server, or if you have more than one Backup Server running on your system, modify the *pname* for server SYB_BACKUP with **sp_addserver** so that Adaptive Server can communicate with Backup Server for database dumps and loads.
- If you specify an *lname*, *pname* and *class* that already exist in *sysservers*, **sp_addserver** prints an error message and does not update *sysservers*.
- Use **sp_serveroption** to set or clear server options.
- For information on using Component Integration Services, see the *Component Integration Services User's Guide*.

Permissions

Only a System Security Officer can execute **sp_addserver**.

Tables Used

master.dbo.sysservers, sysobjects

See Also

System procedures	**sp_addremotelogin**, **sp_dropremotelogin**, **sp_dropserver**, **sp_helpremotelogin**, **sp_helpserver**, **sp_serveroption**

sp_addthreshold

Function

Creates a threshold to monitor space on a database segment. When free space on the segment falls below the specified level, Adaptive Server executes the associated stored procedure.

Syntax

sp_addthreshold *dbname*, *segname*, *free_space*, *proc_name*

Parameters

dbname – is the database for which to add the threshold. This must be the name of the current database.

segname – is the segment for which to monitor free space. Use quotes when specifying the "default" segment.

free_space – is the number of free pages at which the threshold is crossed. When free space in the segment falls below this level, Adaptive Server executes the associated stored procedure.

proc_name – is the stored procedure to be executed when the amount of free space on *segname* drops below *free_space*. The procedure can be located in any database on the current Adaptive Server or on an Open Server. Thresholds cannot execute procedures on remote Adaptive Servers.

Examples

1. **sp_addthreshold mydb, segment1, 200, pr_warning**

   Creates a threshold for *segment1*. When the free space on *segment1* drops below 200 pages, Adaptive Server executes the procedure **pr_warning**.

2. **sp_addthreshold userdb, user_data, 100,**
   "o_server...mail_me"

   Creates a threshold for the *user_data* segment. When the free space on *user_data* falls below 100 pages, Adaptive Server executes a remote procedure call to the Open Server **mail_me** procedure.

```
3. pubs2..sp_addthreshold pubs2, indexes, 100,
   pr_warning
```

Creates a threshold on the *indexes* segment of the *pubs2* database. You can issue this command from any database.

Comments

- For more information about using thresholds, see the *System Administration Guide.*

Crossing a Threshold

- When a threshold is crossed, Adaptive Server executes the associated stored procedure. Adaptive Server uses the following search path for the threshold procedure:
  - If the procedure name does not specify a database, Adaptive Server looks in the database in which the threshold was crossed.
  - If the procedure is not found in this database, and the procedure name begins with "sp_", Adaptive Server looks in the *sybsystemprocs* database.

  If the procedure is not found in either database, Adaptive Server sends an error message to the error log.
- Adaptive Server uses a **hysteresis value**, the global variable *@@thresh_hysteresis*, to determine how sensitive thresholds are to variations in free space. Once a threshold executes its procedure, it is deactivated. The threshold remains inactive until the amount of free space in the segment rises to *@@thresh_hysteresis* pages above the threshold. This prevents thresholds from executing their procedures repeatedly in response to minor fluctuations in free space.

The Last-Chance Threshold

- By default, Adaptive Server monitors the free space on the segment where the log resides and executes **sp_thresholdaction** when the amount of free space is less than that required to permit a successful dump of the transaction log. This amount of free space, called the **last-chance threshold**, is calculated by Adaptive Server and cannot be changed by users.
- If the last-chance threshold is crossed before a transaction is logged, Adaptive Server suspends the transaction until log space is freed. Use **sp_dboption** to change this behavior for a particular

database. **sp_dboption "abort tran on log full", true** causes Adaptive Server to roll back all transactions that have not yet been logged when the last-chance threshold is crossed.

Creating Additional Thresholds

- Each database can have up to 256 thresholds, including the last-chance threshold.
- When you add a threshold, it must be at least 2 times *@@thresh_hysteresis* pages from the closest threshold.

Creating Threshold Procedures

- Any user with **create procedure** permission can create a threshold procedure in a database. Usually, a System Administrator creates **sp_thresholdaction** in the *sybsystemprocs* database, and the Database Owners create threshold procedures in user databases.
- **sp_addthreshold** does not verify that the specified procedure exists. It is possible to add a threshold before creating the procedure it executes.
- **sp_addthreshold** checks to ensure that the user adding the threshold procedure has been directly granted the “sa_role”. All system roles active when the threshold procedure is created are entered in *systhresholds* as valid roles for the user writing the procedure. However, only directly granted system roles are activated when the threshold fires. Indirectly granted system roles and user-defined roles are not activated.
- Adaptive Server passes four parameters to a threshold procedure:
  - *@dbname*, *varchar*(30), which identifies the database
  - *@segmentname*, *varchar*(30), which identifies the segment
  - *@space_left*, *int*, which indicates the number of free pages associated with the threshold
  - *@status*, *int*, which has a value of 1 for last-chance thresholds and 0 for other thresholds

  These parameters are passed by position rather than by name; your threshold procedure can use other names for them, but it must declare them in the order shown and with the correct datatypes.
- It is not necessary to create a different procedure for each threshold. To minimize maintenance, you can create a single

threshold procedure in the *sybsystemprocs* database that is executed for all thresholds in Adaptive Server.

- Include **print** and **raiserror** statements in the threshold procedure to send output to the error log.

Executing Threshold Procedures

- Tasks initiated when a threshold is crossed execute as background tasks. These tasks do not have an associated terminal or user session. If you execute **sp_who** while these tasks are running, the *status* column shows "background".
- Adaptive Server executes the threshold procedure with the permissions the user had at the time he or she added the threshold, minus any permissions that have since been revoked.
- Each threshold procedure uses one user connection, for as long as it takes for the procedure to execute.

Changing or Deleting Thresholds

- Use **sp_helpthreshold** for information about existing thresholds.
- Use **sp_modifythreshold** to associate a threshold with a new threshold procedure, free-space value, or segment. (You cannot change the free-space value or segment name associated with the last-chance threshold.)

  Each time a user modifies a threshold, that user becomes the threshold owner. When the threshold is crossed, Adaptive Server executes the threshold with the permissions the owner had at the time he or she modified the threshold, minus any permissions that have since been revoked.
- Use **sp_dropthreshold** to drop a threshold from a segment.

Disabling Free-Space Accounting

- Use the **no free space acctg** option of **sp_dboption** to disable free-space accounting on non-log segments.
- You cannot disable free-space accounting on log segments.

◆ ***WARNING!***

System procedures cannot provide accurate information about space allocation when free-space accounting is disabled.

Creating Last-Chance Thresholds for Pre-Release 11.0 Databases

- Databases do not automatically acquire a last-chance threshold when upgraded from a release prior to 10.0. Use the **lct_admin** system function to create a last-chance threshold in a pre-10.0 database upgraded to the current release.
- Only databases that store their logs on a separate segment can have a last-chance threshold. Use **sp_logdevice** to move the transaction log to a separate device.

Permissions

Only the Database Owner or a System Administrator can execute **sp_addthreshold.**

Tables Used

master.dbo.sysusages, *sysobjects*, *syssegments*, *systhresholds*

See Also

Commands	**create procedure**, **dump transaction**
Functions	**lct_admin**
System procedures	**sp_dboption**, **sp_dropthreshold**, **sp_helpthreshold**, **sp_modifythreshold**, **sp_thresholdaction**

sp_add_time_range

Function

Adds a named time range to an Adaptive Server.

Syntax

```
sp_add_time_range name, startday, endday,
   starttime, endtime
```

Parameters

name – is the name of the time range. Time range names must be 30 characters or fewer. The name cannot already exist in the *systimeranges* system table of the *master* database.

startday – is the day of the week on which the time range begins. This must be the full weekday name for the default server language, as stored in the *syslanguages* system table of the *master* database.

endday – is the day of the week on which the time range ends. This must be the full weekday name for the default server language, as stored in the *syslanguages* system table of the *master* database. The *endday* can fall either earlier or later in the week than the *startday* or can be the same day as the *startday*.

starttime – is the time of day when the time range begins. Specify the *starttime* in terms of a 24-hour clock, with a value between "00:00" (midnight) and "23:59" (11:59 p.m.). Use the following form:

"*HH*:*MM*"

endtime – is the time of day when the time range ends. Specify the *endtime* in terms of a 24-hour clock, with a value between "00:00" (midnight) and "23:59" (11:59 p.m.). Use the following form:

"*HH*:*MM*"

➤ ***Note***

To create a time range that spans the entire day, specify both a start time and an end time of "00:00".

The *endtime* must occur later in the day than the *starttime*, unless *endtime* is "00:00".

Examples

1. **`sp_add_time_range business_hours, monday, Friday, "09:00", "17:00"`**

   Creates the *business_hours* time range, which is active Monday through Friday, from 9:00 a.m. to 5:00 p.m.

2. **`sp_add_time_range before_hours, Monday, Friday, "00:00", "09:00"`**

   `sp_add_time_range after_hours, Monday, Friday, "18:00", "00:00"`

   Creates two time ranges, *before_hours* and *after_hours*, that, together, span all non-business hours Monday through Friday. The *before_hours* time range covers the period from 12:00 midnight to 9:00 a.m., Monday through Friday. The *after_hours* time range covers the period from 6:00 p.m. through 12:00 midnight, Monday through Friday.

3. **`sp_add_time_range weekends, Saturday, Sunday, "00:00", "00:00"`**

   Creates the *weekends* time range, which is 12:00 midnight Saturday to 12:00 midnight Sunday.

4. **`sp_add_time_range Fri_thru_Mon, Friday, Monday, "09:00", "17:00"`**

   Creates the *Fri_thru_Mon* time range, which is 9:00 a.m. to 5:00 p.m., Friday, Saturday, Sunday, and Monday.

5. **`sp_add_time_range Wednesday_night, Wednesday, Wednesday, "17:00", "00:00"`**

   Creates the *Wednesday_night* time range, which is Wednesday from 5:00 p.m. to 12:00 midnight.

Comments

- Adaptive Server includes one named time range, the “at all times” time range. This time range covers all times, from the first day through the last of the week, from 00:00 through 23:59. It cannot be modified or deleted.
- Adaptive Server generates a unique ID number for each named time range and inserts it into the *systimeranges* system table,
- When storing a time range in the *systimeranges* system table, Adaptive Server converts its *startday* and *endday* values into integers. For servers with a default language of us_english, the week begins on Monday (day 1) and ends on Sunday (day 7).

- It is possible to create a time range that overlaps with one or more other time ranges.
- Range days are contiguous, so the days of the week can wrap around the end to the beginning of the week. In other words, Sunday and Monday are contiguous days, as are Tuesday and Wednesday.
- The active time ranges are bound to a session at the beginning of each query batch. A change in the server's active time ranges due to a change in actual time has no effect on a session during the processing of a query batch. In other words, if a resource limit restricts a query batch during a given time range but a query batch begins before that time range becomes active, the query batch that is already running is not affected by the resource limit.
- The addition, modification, and deletion of time ranges using the system procedures does not affect the active time ranges for sessions currently in progress.
- If a resource limit has a transaction as its scope, and a change occurs in the server's active time ranges while a transaction is running, the newly active time range does not affect the transaction currently in progress.
- Changes to a resource limit that has a transaction as its scope does not affect any transactions currently in progress.
- For more information on time ranges, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_add_time_range**.

Tables Used

master..systimeranges, master..syslanguages

See Also

System procedures	**sp_add_resource_limit**, **sp_drop_time_range**, **sp_modify_time_range**

sp_addtype

Function

Creates a user-defined datatype.

Syntax

```
sp_addtype typename,
   phystype [(length) | (precision [, scale])]
   [, "identity" | nulltype]
```

Parameters

typename – is the name of the user-defined datatype. Type names must conform to the rules for identifiers and must be unique in each database.

phystype – is the physical or Adaptive Server-supplied datatype on which to base the user-defined datatype. You can specify any Adaptive Server datatype except *timestamp*.

The *char*, *varchar*, *nchar*, *nvarchar*, *binary*, and *varbinary* datatypes expect a *length* in parentheses. If you do not supply one, Adaptive Server uses the default length of 1 character.

The *float* datatype expects a binary *precision* in parentheses. If you do not supply one, Adaptive Server uses the default precision for your platform.

The *numeric* and *decimal* datatypes expect a decimal *precision* and *scale*, in parentheses and separated by a comma. If you do not supply them, Adaptive Server uses a default precision of 18 and a scale of 0.

Enclose physical types that include punctuation, such as parentheses or commas, within single or double quotes.

identity – indicates that the user-defined datatype has the IDENTITY property. Enclose the **identity** keyword within single or double quotes. You can specify the IDENTITY property only for *numeric* datatypes with a scale of 0.

IDENTITY columns store sequential numbers, such as invoice numbers or employee numbers, that are generated by Adaptive Server. The value of the IDENTITY column uniquely identifies each row in a table. IDENTITY columns are not updatable and do not allow null values.

nulltype – indicates how the user-defined datatype handles null value entries. Acceptable values for this parameter are **null**, **NULL**, **nonull**, **NONULL**, **"not null"**, and **"NOT NULL"**. Any *nulltype* that includes a blank space must be enclosed in single or double quotes.

If you omit both the IDENTITY property and the *nulltype*, Adaptive Server creates the datatype using the null mode defined for the database. By default, datatypes for which no *nulltype* is specified are created NOT NULL (that is, null values are not allowed and explicit entries are required). For compliance to the SQL standards, use the **sp_dboption** system procedure to set the **allow nulls by default** option to **true**. This changes the database's null mode to NULL.

Examples

1. **`sp_addtype ssn, "varchar(11)"`**

   Creates a user-defined datatype called *ssn* to be used for columns that hold social security numbers. Since the *nulltype* parameter is not specified, Adaptive Server creates the datatype using the database's default null mode. Notice that *varchar*(11) is enclosed in quotation marks, because it contains punctuation (parentheses).

2. **`sp_addtype birthday, "datetime", null`**

   Creates a user-defined datatype called *birthday* that allows null values.

3. **`sp_addtype temp52, "numeric(5,2)"`**

   Creates a user-defined datatype called *temp52* used to store temperatures of up to 5 significant digits with 2 places to the right of the decimal point.

4. **`sp_addtype "row_id", "numeric(10,0)", "identity"`**

   Creates a user-defined datatype called *row_id* with the IDENTITY property, to be used as a unique row identifier. Columns created with this datatype store system-generated values of up to 10 digits in length.

5. **`sp_addtype systype, sysname`**

   Creates a user-defined datatype with an underlying type of *sysname*. Although you cannot use the *sysname* datatype in a **create table**, **alter table**, or **create procedure** statement, you can use a user-defined datatype that is based on *sysname*.

Comments

- **sp_addtype** creates a user-defined datatype and adds it to the *systypes* system table. Once a user-defined datatype is created, you can use it in **create table** and **alter table** statements and bind defaults and rules to it.
- Build each user-defined datatype in terms of one of the Adaptive Server-supplied datatypes, specifying the length or the precision and scale, as appropriate. You cannot override the length, precision, or scale in a **create table** or **alter table** statement.
- A user-defined datatype name must be unique in the database, but user-defined datatypes with different names can have the same definitions.
- If *nchar* or *nvarchar* is specified as the *phystype*, the maximum length of columns created with the new type is the length specified in **sp_addtype** multiplied by the value of *@@ncharsize* at the time the type was added.
- Each system type has a **hierarchy**, stored in the *systypes* system table. User-defined datatypes have the same datatype hierarchy as the physical types on which they are based. In a mixed-mode expression, all types are converted to a common type, the type with the lowest hierarchy.

  Use the following query to list the hierarchy for each system-supplied and user-defined type in your database:

  ```
  select name, hierarchy
  from systypes
  order by hierarchy
  ```

Types with the IDENTITY Property

- If a user-defined datatype is defined with the IDENTITY property, all columns created from it are IDENTITY columns. You can specify either IDENTITY or NOT NULL—or neither one—in the **create** or **alter table** statement. Following are three different ways to create an IDENTITY column from a user-defined datatype with the IDENTITY property:

  ```
  create table new_table (id_col IdentType)
  create table new_table (id_col IdentType identity)
  create table new_table (id_col IdentType not null)
  ```

- When you create a column with the **create table** or **alter table** statement, you can override the null type specified with the **sp_addtype** system procedure:
  - Types specified as NOT NULL can be used to create NULL or IDENTITY columns.
  - Types specified as NULL can be used to create NOT NULL columns, but not to create IDENTITY columns.

➤ ***Note***

If you try to create a null column from an IDENTITY type, the **create** or **alter table** statement fails.

Permissions

Any user can execute **sp_addtype**.

Tables Used

master.dbo.spt_values, *master.dbo.sysdatabases*, *sysobjects*, *systypes*

See Also

Commands	**create default**, **create rule**, **create table**
Datatypes	"User-Defined Datatypes"
System procedures	**sp_bindefault**, **sp_bindrule**, **sp_dboption**, **sp_droptype**, **sp_rename**, **sp_unbindefault**, **sp_unbindrule**

sp_addumpdevice

Function

Adds a dump device to Adaptive Server.

Syntax

```
sp_addumpdevice {"tape" | "disk"}, logicalname,
   physicalname [, tapesize]
```

Parameters

"tape" – for tape drives. Enclose **tape** in quotes.

"disk" – is for a disk or a file device. Enclose **disk** in quotes.

logicalname – is the "logical" dump device name. It must be a valid identifier. Once you add a dump device to *sysdevices*, you can specify its logical name in the **load** and **dump** commands.

physicalname – is the physical name of the device. You can specify either an absolute path name or a relative path name. During dumps and loads, the Backup Server resolves relative path names by looking in Adaptive Server's current working directory. Enclose names containing non-alphanumeric characters in quotation marks. For UNIX platforms, specify a non-rewinding tape device name.

tapesize – is the capacity of the tape dump device, specified in megabytes. OpenVMS systems ignore the *tapesize* parameter if specified. Other platforms require this parameter for tape devices but ignore it for disk devices. The *tapesize* should be at least five database pages (each page requires 2048 bytes). Sybase recommends that you specify a capacity that is slightly below the rated capacity for your device.

Examples

1. ```
   sp_addumpdevice "tape", mytapedump, "/dev/nrmt8",
   40
   ```

   Adds a 40MB tape device. Dump and load commands can reference the device by its physical name, */dev/nrmt8*, or its logical name, *mytapedump*.
   ```

2. **sp_addumpdevice "disk", mydiskdump, "/dev/rxy1d/dump.dat"**

   Adds a disk device named *mydiskdump*. Specify an absolute or relative path name and a file name.

Comments

- **sp_addumpdevice** adds a dump device to the *master.dbo.sysdevices* table. Tape devices are assigned a *cntrltype* of 3; disk devices are assigned a *cntrltype* of 2.
- To use an operating system file as a dump device, specify a device of type **disk** and an absolute or relative path name for the *physicalname*. Omit the *tapesize* parameter. If you specify a relative path name, dumps are made to—or loaded from—the current Adaptive Server working directory at the time the dump or load command executes.
- Ownership and permission problems can interfere with the use of disk or file dump devices. **sp_addumpdevice** adds the device to the *sysdevices* table, but does not guarantee that you can create a file as a dump device or that users can dump to a particular device.
- The **with capacity** = *megabytes* clause of the **dump database** and **dump transaction** commands can override the *tapesize* specified with **sp_addumpdevice**. On platforms that do not reliably detect the end-of-tape marker, the Backup Server issues a volume change request after the specified number of megabytes have been dumped.
- When a dump device fails, use **sp_dropdevice** to drop it from *sysdevices*. After replacing the device, use **sp_addumpdevice** to associate the logical device name with the new physical device. This avoids updating backup scripts and threshold procedures each time a dump device fails.
- To add database devices to *sysdevices*, use the **disk init** command.

Permissions

Only a System Administrator can execute **sp_addumpdevice**.

Tables Used

master.dbo.sysdevices, *sysobjects*

See Also

Commands	**disk init**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_dropdevice**, **sp_helpdevice**

sp_adduser

Function

Adds a new user to the current database.

Syntax

```
sp_adduser loginame [, name_in_db [, grpname]]
```

Parameters

loginame – is the user's name in *master.dbo.syslogins*.

name_in_db – is a new name for the user in the current database.

grpname – adds the user to an existing group in the database.

Examples

1. `sp_adduser margaret`

   Adds "margaret" to the database. Her database user name is the same as her Adaptive Server login name, and she belongs to the default group, "public".

2. `sp_adduser haroldq, harold, fort_mudge`

   Adds "haroldq" to the database. When "haroldq" uses the current database, his name is "harold." He belongs to the *fort_mudge* group, as well as to the default group "public".

Comments

- The Database Owner executes **sp_adduser** to add a user name to the *sysusers* table of the current database, enabling the user to access the current database under his or her own name.
- Specifying a *name_in_db* parameter gives the new user a name in the database that is different from his or her login name in Adaptive Server. The ability to assign a user a different name is provided as a convenience. It is not an alias, as provided by **sp_addalias**, since it is not mapped to the identity and privileges of another user.
- A user and a group cannot have the same name.
- A user can be a member of only one group other than the default group, "public". Every user is a member of the default group, "public". Use **sp_changegroup** to change a user's group.

- In order to access a database, a user must either be listed in *sysusers* (with **sp_adduser**) or mapped to another user in *sysalternates* (with **sp_addalias**), or there must be a "guest" entry in *sysusers*.

Permissions

Only the Database Owner, a System Administrator, or a System Security Officer can execute **sp_adduser**.

Tables Used

master.dbo.syslogins, *master.dbo.syssrvroles*, *sysalternates*, *sysobjects*, *sysusers*

See Also

Commands	**grant**, **revoke**, **use**
System procedures	**sp_addalias**, **sp_addgroup**, **sp_changegroup**, **sp_dropalias**, **sp_dropgroup**, **sp_helpuser**

sp_altermessage

Function

Enables and disables the logging of a system-defined or user-defined message in the Adaptive Server error log.

Syntax

```
sp_altermessage message_id, parameter, parameter_value
```

Parameters

message_id – is the message number of the message to be altered. This is the number of the message as it is recorded in the *error* column in the *sysmessages* or *sysusermessages* system table.

parameter – is the message parameter to be altered. The maximum length is 255 bytes. The only valid parameter is **with_log**.

parameter_value – is the new value for the parameter specified in *parameter*. The maximum length is 255 bytes. Values are **true** and **false**.

Examples

```
1. sp_altermessage 2000, 'with_log', 'TRUE'
```

Specifies that message number 2000 in *sysmessages* should be logged in the Adaptive Server error log and also in the Windows NT Event Log (if logging is enabled).

Comments

- If the *parameter_value* is **true**, the specified message is always logged. If it is **false**, the default logging behavior is used; the message may or may not be logged, depending on the severity of the error and other factors. Setting the *parameter_value* to **false** produces the same behavior that would occur if **sp_altermessage** had not been called.
- On Windows NT servers, **sp_altermessage** also enables and disables logging in the Windows NT Event Log.

Permissions

Only the Database Owner or a System Administrator can execute **sp_altermessage.**

Tables Used

sysmessages, *sysusermessages*

See Also

System procedures	**sp_addmessage**, **sp_dropmessage**

sp_audit

Function

Allows a System Security Officer to configure auditing options.

Syntax

sp_audit ***option*, *login_name*, *object_name*** **[,*setting*]**

Parameters

option – is the name of the auditing option to set. Table 7-5 lists the valid auditing options.

Table 7-5: Auditing options

Option	Description
adhoc	Allows users to use **sp_addauditrecord** to add their own user-defined audit records to the audit trail
all	Audits all actions performed by a particular user or by users with a particular role **Note:** Auditing all actions does not affect whether users can add ad hoc audit records.
alter	Audits the execution of the **alter table** or **alter database** commands
bcp	Audits the execution of the **bcp in** utility
bind	Audits the execution of **sp_bindefault**, **sp_bindmsg**, and **sp_bindrule** system procedures
cmdtext	Audits all actions of a particular user, or by users with a particular role.
create	Audits the creation of database objects
dbaccess	Audits access to the current database from another database
dbcc	Audits the execution of any **dbcc** command
delete	Audits the deletion of rows from a table or view
disk	Audits the execution of **disk init**, **disk refit**, **disk reinit**, **disk mirror**, **disk unmirror**, and **disk remirror**
drop	Audits the dropping of database objects
dump	Audits the execution of **dump database** or **dump transaction** commands

Table 7-5: Auditing options (continued)

Option	Description
errors	Audits errors, whether fatal or not
exec_procedure	Audits the execution of a stored procedure
exec_trigger	Audits the execution of a trigger
func_dbaccess	Audits access to a database via a Transact-SQL function
func_obj_access	Audits access to a database object via a Transact-SQL function
grant	Audits the execution of the **grant** command
insert	Audits the insertion of rows into a table or view
load	Audits the execution of the **load database** or **load transaction** commands
login	Audits all login attempts into Adaptive Server
logout	Audits all logout attempts from Adaptive Server
reference	Audits references between tables.
revoke	Audits the execution of the **revoke** command
rpc	Audits the execution of remote procedure calls
security	Audits the following security-relevant events: • Starting up or shutting down the server • Activating or deactivating a role • Issuing any of the following commands: **connect** **kill** **online database** **set proxy** **set session authorization** **sp_configure** • Using any of the following functions: **valid_user** **proc_role** (from within a system procedure) • Regenerating the SSO passwords
select	Audits the execution of the **select** command
setuser	Audits the execution of the **setuser** command
table_access	Audits access to any table by a specific user

Table 7-5: Auditing options (continued)

Option	Description
truncate	Audits the execution of the **truncate table** command
unbind	Audits the execution of the **sp_unbindrule**, **sp_unbindmsg**, and **sp_unbindefault** system procedures
update	Audits updates to rows in a table or view
view_access	Audits access to any view by a specific user

login_name – is the name of a specific login to be audited. To audit all logins, specify **all** for the *option* parameter. If you specify **all**, you can set the *login_name* parameter to a specific system role to audit all actions by users with that system role active. You cannot specify a user-defined role for *login_name*. For more information on the *login_name* values that are valid with each *option* value, see the *System Administration Guide*.

object_name – is the name of the object to be audited. Valid values, depending on the value you specified for *option*, are:

- The object name, including the owner's name if you do not own the object. For example, to audit a table named *inventory* that is owned by Joe, you would specify *joe.inventory* for *object_name*.
- **all** for all objects.
- **default table**, **default view**, **default procedure**, or **default trigger** to audit access to any new table, view, procedure, or trigger.

  default table and **default view** are valid values for *object_name* when you specify **delete**, **insert**, **select**, or **update** for the *option* parameter. **default procedure** is valid when you specify the **exec_procedure** option. **default trigger** is valid when you specify the **exec_trigger** option.

For more information about the *object_name* values that are valid with each *option* value, see the *System Administration Guide*.

setting – is the level of auditing. If you do not specify a value for *setting*, Adaptive Server displays the current auditing setting for

the option. Valid values for the *setting* parameter are described in the following table:

setting value	Description
on	Activates auditing for the specified option. Adaptive Server generates audit records for events controlled by this option, whether the event passes or fails permission checks.
off	Deactivates auditing for the specified option.
pass	Activates auditing for events that pass permission checks.
fail	Deactivates auditing for events that fail permission checks.

If you specify **pass** for an option and later specify **fail** for the same option, or vice versa, the result is equivalent to specifying **on**. Adaptive Server generates audit records regardless of whether events pass or fail permission checks. Settings of **on** or **off** apply to all auditing options. Settings of **pass** and **fail** apply to all options except **errors** and **adhoc**. For these options, only **on** or **off** applies. The initial, default value of all options is **off**.

Examples

1. `sp_audit "security", "all", "all", "on"`

   Initiates auditing for security-relevant events. Both successful and failed events are audited.

2. `sp_audit "security", "all", "all"`

   Displays the setting of the **security** auditing option.

3. `sp_audit "create", "all", master, "on"`

   Initiates auditing for the creation of objects in the *master* database, including **create database**.

4. `sp_audit "create", "all", db1, "on"`

   Initiates auditing for the creation of all objects in the *db1* database.

5. `sp_audit "all", "sa_role", "all", "fail"`

   Initiates auditing for all failed executions by a System Administrator.

6. `sp_audit "update", "all", "default table", "on"`

Initiates auditing for all updates to future tables in the current database. For example, if the current database is *utility*, all new tables created in *utility* will be audited for updates. The auditing for existing tables is not affected.

Comments

- **sp_audit** determines what will be audited when auditing is enabled. No actual auditing takes place until you use **sp_configure** to set the **auditing** parameter to **on**. Then, all auditing options that have been configured with **sp_audit** take effect. For more information, see **sp_configure**.
- If you are not the owner of the object being specified, qualify the *object_name* parameter value with the owner's name, in the following format:

  "*ownername.objname*"
- You cannot activate default auditing for the following options in the *tempdb* database:
  - **delete**
  - **insert**
  - **select**
  - **update**
  - **exec_procedure**
  - **exec_trigger**
- Table 7-6 lists the configuration parameters that control auditing.

Table 7-6: Configuration parameters that control auditing

Configuration Parameter	Effect
auditing	Enables or disables auditing for the server.
audit_queue_size	Establishes the size of the audit queue.
current_audit_table	Sets the current audit table. Adaptive Server writes all audit records to that table.
suspend_auditing_when_full	Controls the behavior of the audit process when an audit device becomes full.

The **auditing**, **current_audit_table**, and **suspend_auditing_when_full** configuration parameters are dynamic and take effect immediately. Because **audit_queue_size** affects memory allocation,

the parameter is static and does not take effect until Adaptive Server is restarted.

- For more information about configuring Adaptive Server for auditing, see **sp_configure** in the *System Administration Guide.*

Permissions

Only a System Security Officer can execute **sp_audit.**

Tables Used

master..sysdatabases, *sysobjects*, *sybsecurity..sysauditoptions*, *sybsecurity..sysaudits_01...sybsecurity..sysaudits_08*

See Also

System procedures	**sp_addauditrecord**, **sp_configure**
Utility commands	**bcp**

sp_autoconnect

(Component Integration Services only)

Function

Defines a passthrough connection to a remote server for a specific user, which allows the named user to enter passthrough mode automatically at login.

Syntax

```
sp_autoconnect server, {true|false}
   [, loginame]
```

Parameters

server – is the name of a server to which an automatic passthrough connection is made. *server* must be the name of a remote server already added by **sp_addserver**. This server cannot be the local server.

true | **false** – determines whether the automatic passthrough connection is enabled or disabled for *server*. **true** enables the automatic connection. **false** disables it.

loginame – specifies the name of the user for which automatic connection is required. If no *loginame* is supplied, the autoconnect status is modified for the current user.

Examples

1. `sp_autoconnect SYBASE, true`

   The current user is automatically connected to the server SYBASE the next time that user logs in. The user's connection is placed in passthrough mode.

2. `sp_autoconnect SYBASE, false, steve`

   Disables the autoconnect feature for the user "steve".

Comments

- **sp_autoconnect** defines a passthrough connection to a remote server for a specific user, which allows the named user to enter passthrough mode automatically at login.
- The System Administrator must grant **connect to** permission to the login prior to executing **sp_autoconnect**.

- Use **sp_autoconnect** only when Component Integration Services is installed and configured.
- Do not change the autoconnect status of the "sa" login account.
- Changing the autoconnect status does not occur immediately for users who are currently connected. They must disconnect from the local server, then reconnect before the change is made.
- Use **disconnect** to exit passthrough mode.

Permissions

Only a System Administrator can execute **sp_autoconnect**.

Tables Used

sysservers, *syslogins*

See Also

Commands	**connect to...disconnect**, **grant**
System procedures	**sp_addlogin**, **sp_passthru**, **sp_remotesql**, **sp_addserver**

sp_bindcache

Function

Binds a database, table, index, *text* object, or *image* object to a data cache.

Syntax

```
sp_bindcache cachename, dbname
   [, [ownername.]tablename
   [, indexname | "text only"]]
```

Parameters

cachename – is the name of an active data cache.

dbname – is the name of the database to be bound to the cache or the name of the database containing the table, index, *text* or *image* object to be bound to the cache.

ownername – is the name of the table's owner. If the table is owned by "dbo", the owner name is optional.

tablename – is the name of the table to be bound to the cache, or the name of the table whose index, *text* object, or *image* object is to be bound to the cache.

indexname – is the name of the index to be bound to the cache.

text only – binds *text* or *image* objects to a cache. When this parameter is used, you cannot give an index name at the same time.

Examples

1. `sp_bindcache pub_cache, pubs2, titles`

   Binds the *titles* table to the cache named *pub_cache*.

2. `sp_bindcache pub_ix_cache, pubs2, titles, title_id_cix`

   Binds the clustered index *titles.title_id_cix* to the *pub_ix_cache*.

3. `sp_bindcache tempdb_cache, tempdb`

   Binds *tempdb* to the *tempdb_cache*.

4. `sp_bindcache logcache, pubs2, syslogs`

   Binds the *pubs2* transaction log, *syslogs*, to the cache named *logcache*.

5. **`sp_bindcache pub_cache, pubs2, au_pix, "text only"`**

   Binds the *image* chain for the *au_pix* table to the cache named *pub_cache*.

Comments

- A database or database object can be bound to only one cache. You can bind a database to one cache and bind individual tables, indexes, *text* objects, or *image* objects in the database to other caches. The database binding serves as the default binding for all objects in the database that have no other binding. The data cache hierarchy for a table or index is as follows:
  - If the object is bound to a cache, the object binding is used.
  - If the object is not bound to a cache, but the object's database is bound to a cache, the database binding is used.
  - If neither the object nor its database is bound to a cache, the default data cache is used.
- The cache and the object or database being bound to it must exist before you can execute **sp_bindcache**. Create a cache with **sp_cacheconfig** and restart Adaptive Server before binding objects to the cache.
- Cache bindings take effect immediately, and do not require a restart of the server. When you bind an object to a data cache:
  - Any pages for the object that are currently in memory are cleared.
  - When the object is used in queries, its pages are read into the bound cache.
- You can bind an index to a different cache than the table it references. If you bind a clustered index to a cache, the binding affects only the root and intermediate pages of the index. It does not affect the data pages (which are, by definition, the leaf pages of the index).
- To bind a database, you must be using the *master* database. To bind tables, indexes, *text* objects, or *image* objects, you must be using the database where the objects are stored.
- To bind any system tables in a database, you must be using the database and the database must be in single-user mode. Use the command:

  `sp_dboption` ***`db_name`***`, "single user", true`

  For more information, see **sp_dpoption**.

- You do not have to unbind objects or databases in order to bind them to a different cache. Issuing **sp_bindcache** on an object that is already bound drops the old binding and creates the new one.
- **sp_bindcache** needs to acquire an exclusive table lock when you are binding a table or its indexes to a cache so that no pages can be read while the binding is taking place. If a user holds locks on a table, and you issue **sp_bindcache** on that object, the task doing the binding sleeps until the locks are released.
- When you bind or unbind an object, all stored procedures that reference the object are recompiled the next time they are executed. When you change the binding for a database, all stored procedures that reference objects in the bound database are recompiled the next time they are executed.
- When you drop a table, index, or database, all associated cache bindings are dropped. If you re-create the table, index, or database, you must use **sp_bindcache** again if you want it bound to a cache.
- If a database or a database object is bound to a cache, and the cache is dropped, the cache bindings are marked invalid, but remain stored in the *sysattributes* system table(s). Warnings are printed in the error log when Adaptive Server is restarted. If a cache of the same name is created, the bindings become valid when Adaptive Server is restarted.
- The following procedures provide information about the bindings for their respective objects: **sp_helpdb** for databases, **sp_help** for tables, and **sp_helpindex** for indexes. **sp_helpcache** provides information about all objects bound to a particular cache.
- Use **sp_spaceused** to see the current size of tables and indexes, and **sp_estspace** to estimate the size of tables that you expect to grow. Use **sp_cacheconfig** to see information about cache size and status, and to configure and reconfigure caches.

Restrictions

- The *master* database, the system tables in *master*, and the indexes on the system tables in *master* cannot be bound to a cache. You can bind non-system tables from *master*, and their indexes, to caches.
- You cannot bind a database or an object to a cache if:
  - Isolation level 0 reads are active on the table

  - The task doing the binding currently has a cursor open on the table

- If a cache has the type **log only**, you can bind a *syslogs* table only to that cache. Use **sp_cacheconfig** to see a cache's type.

Permissions

Only a System Administrator can execute **sp_bindcache**.

Tables Used

master..sysattributes, *master..sysdatabases*, *sysindexes*, *sysobjects*

See Also

System procedures	**sp_cacheconfig**, **sp_configure**, **sp_help**, **sp_helpcache**, **sp_helpdb**, **sp_helpindex**, **sp_poolconfig**, **sp_unbindcache**, **sp_unbindcache_all**

sp_bindefault

Function

Binds a user-defined default to a column or user-defined datatype.

Syntax

```
sp_bindefault defname, objname [, futureonly]
```

Parameters

defname – is the name of a default created with **create default** statements to bind to specific columns or user-defined datatypes.

objname – is the name of the table and column, or user-defined datatype, to which the default is to be bound. If the *objname* parameter is not of the form *"table.column"*, it is assumed to be a user-defined datatype. If the object name includes embedded blanks or punctuation, or is a reserved word, enclose it in quotation marks.

Existing columns of the user-defined datatype inherit the default *defname*, unless you specify **futureonly.**

futureonly – prevents existing columns of a user-defined datatype from acquiring the new default. This parameter is optional when you are binding a default to a user-defined datatype. It is never used to bind a default to a column.

Examples

1. `sp_bindefault today, "employees.startdate"`

   Assuming that a default named *today* has been defined in the current database with **create default**, this command binds it to the *startdate* column of the *employees* table. Each new row added to the *employees* table has the value of the *today* default in the *startdate* column, unless another value is supplied.

2. `sp_bindefault def_ssn, ssn`

   Assuming that a default named *def_ssn* and a user-defined datatype named *ssn* exist, this command binds *def_ssn* to *ssn*. The default is inherited by all columns that are assigned the user-defined datatype *ssn* when a table is created. Existing columns of type *ssn* also inherit the default *def_ssn*, unless you specify **futureonly** (which prevents existing columns of that user-defined datatype from inheriting the default), or unless the column's

default has previously been changed (in which case the changed default is maintained).

3. `sp_bindefault def_ssn, ssn, futureonly`

   Binds the default *def_ssn* to the user-defined datatype *ssn*. Because the **futureonly** parameter is included, no existing columns of type *ssn* are affected.

Comments

- You can create column defaults in two ways: by declaring the default as a column constraint in the **create table** or **alter table** statement or by creating the default using the **create default** statement and binding it to a column using **sp_bindefault**. Using **create default**, you can bind that default to more than one column in the database.
- You cannot bind a default to an Adaptive Server-supplied datatype.
- You cannot bind a default to a system table.
- Defaults bound to a column or user-defined datatype with the IDENTITY property have no effect on column values. Each time you insert a row into the table, Adaptive Server assigns the next sequential number to the IDENTITY column.
- If binding a default to a column, give the *objname* argument in the form "*table.column*". Any other format is assumed to be the name of a user-defined datatype.
- If a default already exists on a column, you must remove it before binding a new default. Use **sp_unbindefault** to remove defaults created with **sp_bindefault**. To remove defaults created with **create table** or **alter table**, use **alter table** to replace the default with NULL.
- Existing columns of the user-defined datatype inherit the new default unless you specify **futureonly**. New columns of the user-defined datatype always inherit the default. Binding a default to a user-defined datatype overrides defaults bound to columns of that type; to restore column bindings, unbind and rebind the column default.
- Statements that use a default cannot be in the same batch as their **sp_bindefault** statement.

Permissions

Only the object owner can execute **sp_bindefault**.

Tables Used

syscolumns, *sysobjects*, *sysprocedures*, *systypes*

See Also

Commands	**create default**, **create table**, **drop default**
System procedures	**sp_unbindefault**

sp_bindexeclass

Function

Associates an execution class with a client application, login, or stored procedure.

Syntax

```
sp_bindexeclass "object_name", "object_type",
   "scope", "classname"
```

Parameters

object_name – is the name of the client application, login, or stored procedure to be associated with the execution class, *classname*.

object_type – identifies the type of *object_name*. Use **ap** for application, **lg** for login, or **pr** for stored procedure.

scope – is the name of a client application or login, or it can be NULL for **ap** and **lg** objects. It is the name of the stored procedure owner (user name) for objects. When the object with *object_name* interacts with the application or login, *classname* attributes apply for the scope you set.

classname – specifies the type of class to associate with *object_name*. Values are:

- *EC1*, *EC2*, or *EC3*
- The name of a user-defined execution class
- *ANYENGINE*

Examples

1. `sp_bindexeclass 'isql', 'ap', NULL, 'EC3'`

   This statement specifies that Transact-SQL applications will execute with *EC3* attributes for any login or application process (because the value of *scope* is NULL) that invokes **isql**, unless the login or application is bound to a higher execution class.

2. `sp_bindexeclass 'sa', 'lg', 'isql', 'EC1'`

   This statement specifies that when a login with the System Administrator role executes Transact-SQL applications, the login process executes with *EC1* attributes. If you have already executed the statement in the first example, then any other login

or client application that invokes **isql** will execute with *EC3* attributes.

1. `sp_bindexeclass 'my_proc', 'PR', 'kundu', 'EC3'`

   This statement assigns *EC3* attributes to the stored procedure named *my_proc* owned by user *kundu*.

Comments

- **sp_bindexeclass** associates an execution class with a client application, login, or stored procedure. Create execution classes with **sp_addexeclass**.
- When *scope* is NULL, *object_name* has no scope. *classname's* execution attributes apply to all of its interactions. For example, if *object_name* is an application name, the attributes apply to any login process that invokes the application. If *object_name* is a login name, the attributes apply to a particular login process for any application invoked by the login process.
- When binding a stored procedure to an execution class, you must use the name of the stored procedure owner (user name) for the *scope* parameter. This narrows the identity of a stored procedure when there are multiple invocations of it in the same database.
- Due to precedence and scoping rules, the execution class being bound may or may not have been in effect for the object called *object_name*. The object automatically binds itself to another execution class, depending on other binding specifications, precedence, and scoping rules. If no other binding is applicable, the object binds to the default execution class, *EC2*.
- Binding fails when you attempt to bind an active process to an engine group with no online engines.
- Adaptive Server creates a row in the *sysattributes* table containing the object ID and user ID in the row that stores data for the binding.
- A stored procedure must exist before it can be bound.
- Stored procedure bindings must be done in the database in which the stored procedure resides. Therefore, when binding system procedures, execute **sp_bindexeclass** from within the *sybsystemprocs* database.
- Only the "priority attribute" of the execution class is used when you bind the class to a stored procedure.

- The name of the owner of a stored procedure must be supplied as the *scope* parameter when you are binding a stored procedure to an execution class. This helps to uniquely identify a stored procedure when multiple stored procedures with the same name (but different owners) exist in the database.

Permissions

Only a System Administrator can execute **sp_bindexeclass**.

Tables Used

sysattributes, syslogins

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**
Utility	**isql**

sp_bindmsg

Function

Binds a user message to a referential integrity constraint or check constraint.

Syntax

sp_bindmsg *constrname*, *msgid*

Parameters

constrname – is the name of the integrity constraint to which you are binding a message. Use the **constraint** clause of the **create table** command, or the **add constraint** clause of the **alter table** command to create and name constraints.

msgid – is the number of the user message to be bound to an integrity constraint. The message must exist in the *sysusermessages* table in the local database prior to calling **sp_bindmsg**.

Examples

1. sp_bindmsg positive_balance, 20100

Binds user message number 20100 to the *positive_balance* constraint.

Comments

- **sp_bindmsg** binds a user message to an integrity constraint by adding the message number to the constraint row in the *sysconstraints* table.
- Only one message can be bound to a constraint. To change the message for a constraint, just bind a new message. The new message number replaces the old message number in the *sysconstraints* table.
- You cannot bind a message to a unique constraint because a unique constraint does not have a constraint row in *sysconstraints* (a unique constraint is a unique index).
- Use the **sp_addmessage** procedure to insert user messages into the *sysusermessages* table.
- The **sp_getmessage** procedure retrieves message text from the *sysusermessages* table.

- **sp_help** *tablename* displays all constraint names declared on *tablename*.

Permissions

Only the object owner can execute **sp_bindmsg**.

Tables Used

sysconstraints, *sysobjects*, *sysusermessages*

See Also

Commands	**alter table**, **create table**
System procedures	**sp_addmessage**, **sp_getmessage**, **sp_unbindmsg**

sp_bindrule

Function

Binds a rule to a column or user-defined datatype.

Syntax

```
sp_bindrule rulename, objname [, futureonly]
```

Parameters

rulename – is the name of a rule. Create rules with **create rule** statements and bind rules to specific columns or user-defined datatypes with **sp_bindrule**.

objname – is the name of the table and column, or user-defined datatype, to which the rule is to be bound. If *objname* is not of the form *"table.column"*, it is assumed to be a user-defined datatype. If the object name has embedded blanks or punctuation, or is a reserved word, enclose it in quotation marks.

futureonly – prevents existing columns of a user-defined datatype from inheriting the new rule. This parameter is optional when you bind a rule to a user-defined datatype. It is meaningless when you bind a rule to a column.

Examples

1. `sp_bindrule today, "employees.startdate"`

   Assuming that a rule named *today* has been created in the current database with **create rule**, this command binds it to the *startdate* column of the *employees* table. When a row is added to *employees*, the data for the *startdate* column is checked against the rule *today*.

2. `sp_bindrule rule_ssn, ssn`

   Assuming the existence of a rule named *rule_ssn* and a user-defined datatype named *ssn*, this command binds *rule_ssn* to *ssn*. In a **create table** statement, columns of type *ssn* inherit the rule *rule_ssn*. Existing columns of type *ssn* also inherit the rule *rule_ssn*, unless *ssn*'s rule was previously changed (in which case the changed rule is maintained in the future only).

3. `sp_bindrule rule_ssn, ssn, futureonly`

   The rule *rule_ssn* is bound to the user-defined datatype *ssn*, but no existing columns of type *ssn* are affected. **futureonly** prevents existing columns of type *ssn* from inheriting the rule.

Comments

- Create a rule using the **create rule** statement. Then execute **sp_bindrule** to bind it to a column or user-defined datatype in the current database.
- Rules are enforced when an **insert** is attempted, not when **sp_bindrule** is executed. You can bind a character rule to a column with an exact or approximate numeric datatype, even though such an **insert** is illegal.
- You cannot use **sp_bindrule** to bind a check constraint for a column in a **create table** statement.
- You cannot bind a rule to an Adaptive Server-supplied datatype or to a *text* or an *image* column.
- You cannot bind a rule to a system table.
- If you are binding to a column, the *objname* argument must be of the form *"table.column"*. Any other format is assumed to be the name of a user-defined datatype.
- Statements that use a rule cannot be in the same batch as their **sp_bindrule** statement.
- You can bind a rule to a column or user-defined datatype without unbinding an existing rule. Rules bound to columns always take precedence over rules bound to user-defined datatypes. Binding a rule to a column will replace a rule bound to the user-defined datatype of that column, but binding a rule to a datatype will not replace a rule bound to a column of that user-defined datatype. Table 7-7 indicates the precedence when binding rules to columns and user-defined datatypes where rules already exist:

Table 7-7: Precedence of new and old bound rules

	Old Rule Bound To:	
New Rule Bound To:	**User-Defined Datatype**	**Column**
user-defined datatype	Replaces old rule	No change
column	Replaces old rule	Replaces old rule

- Existing columns of the user-defined datatype inherit the new rule unless their rule was previously changed, or the value of the optional third parameter is **futureonly**. New columns of the user-defined datatype always inherit the rule.

Permissions

Only the object owner can execute **sp_bindrule**.

Tables Used

syscolumns, *sysconstraints*, *sysobjects*, *sysprocedures*, *systypes*

See Also

Commands	**create rule**, **drop rule**
System procedures	**sp_unbindrule**

sp_cacheconfig

Function

Creates, configures, reconfigures, and drops data caches, and provides information about them.

Syntax

```
sp_cacheconfig [cachename [ ,"cache_size[P|K|M|G]" ]
   [,logonly | mixed ] [,strict | relaxed ] ]
   [, "cache_partition=[1|2|4|8|16|32|64]"]
```

Parameters

cachename – is the name of the data cache to be created or configured. Cache names must be unique, and can be up to 30 characters long. A cache name does not have to be a valid Adaptive Server identifier, that is, it can contain spaces and other special characters.

cache_size – is the size of the data cache to be created or, if the cache already exists, the new size of the data cache. The minimum size of a cache is 512K. Size units can be specified with **P** for pages, **K** for kilobytes, **M** for megabytes, or **G** for gigabytes. The default is **K**. For megabytes and gigabytes, you can specify floating-point values.

logonly | mixed – specifies the type of cache.

strict | relaxed – specifies the cache replacement policy.

cache_partition - specifies the number of partitions to create in the cache. Each pool in the cache must be at least 512K.

Examples

1. **`sp_cacheconfig pub_cache, "10M"`**

   Creates the data cache *pub_cache* with 10MB of space. All space is in the default 2K memory pool.

2. **`sp_cacheconfig pub_cache`**

   Reports the current configuration of *pub_cache* and any memory pools in the cache.

3. **`sp_cacheconfig pub_cache, "0"`**

   Drops *pub_cache* at the next start of Adaptive Server.

4. `sp_cacheconfig pub_log_cache, "2000K", logonly`

   Creates *pub_log_cache* and sets its type to **logonly** in a single step.

5. `sp_cacheconfig pub_log_cache, "2000K"`
   `sp_cacheconfig pub_log_cache, logonly`

   The first command creates the cache *pub_log_cache* with the default type **mixed**. The second command changes its status to **logonly**. The resulting configuration is the same as that in example 4.

6. `sp_cacheconfig 'newcache', '50M', mixed, strict, "cache_partition=2"`

   Creates a cache and sets the size, type, replacement policy and number of cache partitions.

Comments

- Creating data caches divides Adaptive Server's single default data cache into smaller caches. You can then configure pools within a data cache to allow Adaptive Server to perform large I/O using **sp_poolconfig**. You can bind tables, indexes, databases, and *text* or *image* chains to a specific cache using **sp_bindcache**.
- When you first create a data cache:
  - All space is allocated to the 2K memory pool.
  - The default type is **mixed**.
- Figure 7-1 shows a data cache with two user-defined data caches configured and the following pools:
  - The default data cache with a 2K pool and a 16K pool
  - A user cache with a 2K pool and a 16K pool
  - A log cache with a 2K pool and a 4K pool

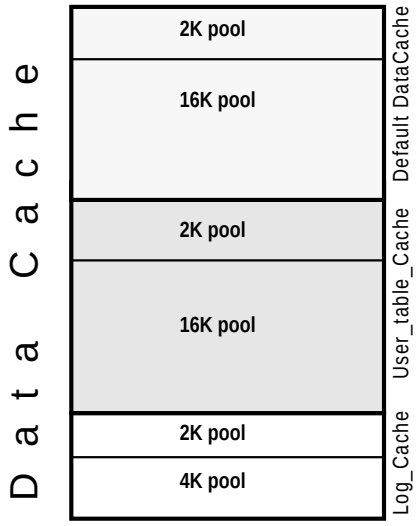

Figure 7-1: Data cache with default and user-defined caches

- Creating, dropping, and changing the replacement policy or number of partitions require a restart of Adaptive Server for the configuration to take effect. You cannot configure pools or bind objects to caches until the cache is active, that is, until the server has been restarted.

  Other changes to data caches take effect without a restart, including changing the type, creating, dropping, and resizing memory pools with **sp_poolconfig**, changing the wash percentage of the pools, and binding and unbinding objects.
- The default data cache must always have the type **default**, and no other cache can have the type **default**.
- The Adaptive Server housekeeper task does not do any buffer washing in caches with a type of **logonly** or in caches with a relaxed LRU replacement policy.
- The following commands perform only 2K I/O: **disk init**, some **dbcc** commands, and **drop table**. The **dbcc checkdb** and **dbcc checktable** commands can perform large I/O for tables, but perform 2K I/O

on indexes. Table 7-8 shows cache usage, depending on the binding of the database or object.

Table 7-8: Cache usage for Transact-SQL commands

Command	Database Bound	Table or Index Is Bound	Database or Object Not Bound
create index	Bound cache	N / A	Default data cache
disk init	N / A	N / A	Default data cache
dbcc checkdb	Bound cache	N / A	Default data cache
dbcc checktable, **indexalloc**, **tablealloc**	Bound cache	Bound cache	Default data cache
drop table	Bound cache	Bound cache	Default data cache

- Recovery uses only the 2K pool of the default data cache. All pages for all transactions that must be rolled back or rolled forward are read into and changed in this pool. Be sure that your default 2K pool is large enough for these transactions.
- When you use **sp_cacheconfig** with no parameters, it reports information about all of the caches on the server. If you specify only a cache name, it reports information about only the specified cache. If you use a fragment of a cache name, it reports information for all names matching "%*fragment*%".

  All reports include a block of information that reports information about caches, and a separate block of data for each cache that provides information about the pools within the cache.

  The output below shows the configuration for:

  - The default data cache with two pools: a 2K pool and a 16K pool. The default data cache has 2 partitions.
  - *pubs_cache* with two pools: 2K and 16K
  - *pubs_log*, with the type set to **logonly** and cache replacement policy set to **relaxed**, with a 2K pool and a 4K pool

```
Cache Name                Status    Type     Config Value Run Value
 ------------------------ --------- -------- ------------ ------------
 default data cache        Active    Default       0.00 Mb     26.09 Mb
 pubs_cache                Active    Mixed        10.00 Mb     10.00 Mb
 pubs_log                  Active    Log Only      2.40 Mb      2.40 Mb
                                                ------------ ------------
                                     Total       12.40 Mb     38.49 Mb
===========================================================================
Cache: default data cache,   Status: Active,   Type: Default
      Config Size: 0.00 Mb,   Run Size: 26.09 Mb
      Config Replacement: strict LRU,   Run Replacement: strict LRU
      Config Partition:            2,   Run Partition:            2
 IO Size  Wash Size Config Size  Run Size     APF Percent
 -------- --------- ------------ ------------ -----------
     2 Kb   3704 Kb      0.00 Mb     18.09 Mb     10
    16 Kb   1632 Kb      8.00 Mb      8.00 Mb     10
===========================================================================
Cache: pubs_cache,   Status: Active,   Type: Mixed
      Config Size: 10.00 Mb,   Run Size: 10.00 Mb
      Config Replacement: strict LRU,   Run Replacement: strict LRU
      Config Partition:            1,   Run Partition:            1
 IO Size  Wash Size Config Size  Run Size     APF Percent
 -------- --------- ------------ ------------ -----------
     2 Kb   1228 Kb      0.00 Mb      6.00 Mb     10
    16 Kb    816 Kb      4.00 Mb      4.00 Mb     10
===========================================================================
Cache: pubs_log,   Status: Active,   Type: Log Only
      Config Size: 2.40 Mb,   Run Size: 2.40 Mb
      Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
      Config Partition:            1,   Run Partition:            1
 IO Size  Wash Size Config Size  Run Size     APF Percent
 -------- --------- ------------ ------------ -----------
     2 Kb    206 Kb      0.00 Mb      1.01 Mb     10
    16 Kb    272 Kb      1.40 Mb      1.39 Mb     10
```

Table 7-9 lists the meaning of the columns in the output:

Table 7-9: sp_cacheconfig output

Column	Meaning
Cache Name	The name of the cache.
Status	One of the following: • “Active” • “Pend / Act” • “Pend / Del” These are explained following this table.

Table 7-9: sp_cacheconfig output (continued)

Column	Meaning
Type	"Mixed" or "Log Only" for user-defined caches, "Default" for the default data cache.
I/O Size	The size of I/O for a memory pool. This column is blank on the line that shows that cache configuration.
Wash Size	The size of the wash area for the pool. As pages enter the wash area of the cache, they are written to disk. This column is blank on the line that shows the cache configuration.
Config Value or Config Size	The size that the cache or pool will have after the next time Adaptive Server is restarted. These are the values that take effect the next time Adaptive Server is restarted. If the value is 0, the size has not been explicitly configured, and a default value will be used.
Run Value or Run Size	The size of the cache or pool now in use on Adaptive Server.
Config/ Run Replacement	The cache policy (strict or relaxed) that will be used for the cache after the next restart, and the current replacement policy. These will be different only if the policy has been changed since the last reboot.
Config/Run Partition	The number of cache partitions that will be used for the cache after the next restart, and the current number of partitions. These will be different if **sp_cacheconfig** has been used to change the number of partitions since the last reboot.
APF Percent	The percentage of buffers in the pool that can hold buffers that have been fetched by asynchronous prefetch, but have not been used.
Total	The total size of data cache, if the report covers all caches, or the current size of the particular cache, if you specify a cache name.

The status "Pend" is short for pending. It always occurs in combination with either "Act" for Active or "Del" for Delete. It indicates that a configuration action has taken place, but that the server must be restarted in order for the changes to take effect.

When you first create a new cache, but have not yet restarted Adaptive Server, the status is "Pend/Act", meaning that the cache has just been configured and will be active after a restart. If you set the size of a cache to 0 to delete it, the status changes

from “Active” to “Pend / Del”, meaning that the cache still exists, and still functions, but that it will be deleted at the next restart.

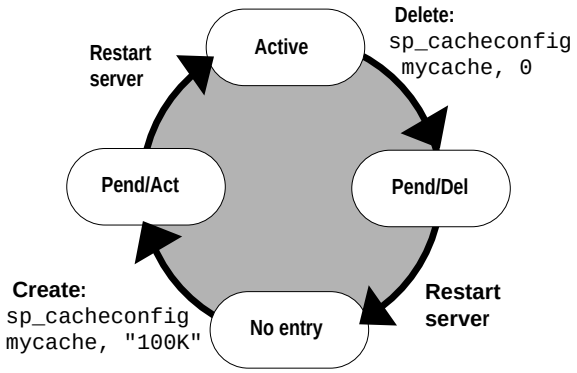

Figure 7-2: Effects of restarts and sp_cacheconfig on cache status

- You can also configure caches and pools by editing the configuration file. For more information, see the *System Administration Guide.*

Data Cache Memory

- When Adaptive Server is first installed, all data cache memory is assigned to the 2K pool of the cache named *default data cache*. The default data cache is used by all objects that are not explicitly bound to a data cache with **sp_bindcache** or whose databases are not bound to a cache.
- When you create data caches, the memory allocation comes from the default data cache. Memory for caches is allocated out of the memory allocated to Adaptive Server with the **total memory** configuration parameter. To increase the amount of space available for caches, increase **total memory**, or decrease other configuration settings that use memory. If you need to decrease the size of **total memory**, the space must be available in the default data cache.

  You cannot reduce the size of the default data cache to less than 512K. In most cases, the default cache should be much larger than the minimum. This cache is used for all objects, including system tables, that are not bound to another cache, and is the only cache used during recovery. For more information, see Chapter 15, “Configuring Data Caches,” in the *System Administration Guide.*

- A data cache requires a small percentage of overhead for structures that manage the cache. All cache overhead is taken from the default data cache. To see the amount of overhead required for a specific size of cache, use **sp_helpcache**, giving the size:

  ```
  sp_helpcache "200M"

  10.38Mb of overhead memory will be needed to
  manage a cache of size 200M
  ```

Changing Existing Caches

- To change the size of an existing cache, specify the cache's name and the new size.
  - If you increase the size of an existing cache, all of the added space is placed in the 2K pool.
  - To reduce the size of an existing cache, all of the space must be available in the 2K pool. You may need to use **sp_poolconfig** to move space from other pools to the 2K pool.
- If you have a database or any nonlog objects bound to a cache, you cannot change its type to **logonly.**

Using Cache Partitions

- Cache partitions can be used to reduce cache spinlock contention without needing to create separate caches and bind database objects to them. For more information on monitoring cache spinlock contention, see Chapter 32, "Memory Use and Performance," in the *Performance and Tuning Guide*.
- You can set the default number of cache partitions for all caches with the configuration parameter **global cache partition number.** See Chapter 15, "Configuring Data Caches," in the *System Administration Guide.*

Dropping Caches

- To drop or delete a data cache, change its size to 0, as shown in example 3. When you set a cache's size to 0, the cache is marked for deletion, but it is not dropped until the next restart of the server. The cache remains active, and all objects that are bound to that cache continue to use it.

  You cannot drop the default data cache.

- If you drop a cache that has objects bound to it, all of the object bindings for the cache are marked invalid the next time you restart Adaptive Server. A message is printed to the error log on restart, giving the database ID, object ID and index ID:

  ```
  00:95/11/05 18:20:39.42 server  Cache binding for
  database '6', object '8', index '0' is being
  marked invalid in Sysattributes.
  ```

  If you subsequently create a cache of the same name, bindings are marked valid when the cache is activated.

Permissions

Only a System Administrator can execute **sp_cacheconfig** to change cache configurations. Any user can execute **sp_cacheconfig** to view cache configurations.

Tables Used

master..sysconfigures, masters..syscurconfigs

See Also

System procedures	**sp_bindcache**, **sp_configure**, **sp_help**, **sp_helpcache**, **sp_helpdb**, **sp_helpindex**, **sp_poolconfig**, **sp_unbindcache**, **sp_unbindcache_all**

sp_cachestrategy

Function

Enables or disables prefetching (large I/O) and MRU cache replacement strategy for a table, index, *text* object, or *image* object.

Syntax

```
sp_cachestrategy dbname, [ownername.]tablename
   [, indexname | "text only" | "table only"
   [, { prefetch | mru }, { "on" | "off"}]]
```

Parameters

dbname – is the name of the database where the object is stored.

ownername – is the name of the table's owner. If the table is owned by "dbo", the owner name is optional.

tablename – is the name of the table.

indexname – is the name of the index on the table.

text only – changes the cache strategy for a *text* or *image* object.

table only – changes the cache strategy for a table.

prefetch | **mru** – is **prefetch** or **mru**, and specifies which setting to change.

on | off – specifies the setting, **"on"** or **"off"**, enclosed in quotes.

Examples

1. **sp_cachestrategy pubs2, titles**

   ```
   object name        index name    large IO MRU
   ------------------ ------------- -------- --------
   dbo.titles           titleidind     ON        ON
   ```

   Displays information about cache strategies for the *titles* table.

2. **sp_cachestrategy pubs2, titles, titleind**

   Displays information about cache strategies for the *titleind* index.

3. **sp_cachestrategy pubs2, titles, titleind, prefetch, "off"**

   Disables prefetch on the *titleind* index of the *titles* table.

4. **sp_cachestrategy pubs2, authors, "table only", mru, "on"**

   Reenables MRU replacement strategy on the *authors* table.

5. **sp_cachestrategy pubs2, blurbs, "text only", prefetch, "on"**

   Reenables prefetching on the text pages of the *blurbs* table.

Comments

- If memory pools for large I/O are configured for the cache used by a table or an index, the optimizer can choose to prefetch data or index pages by performing large I/Os of up to eight data pages at a time. This **prefetch** strategy can be used on the data pages of a table or on the leaf-level pages of a nonclustered index. By default, prefetching is enabled for all tables, indexes, and *text* or *image* objects. Setting the **prefetch** option to "**off**" disables prefetch for the specified object.
- The optimizer can choose to use **MRU replacement strategy** to fetch and discard buffers in cache for table scans and index scans for I/O of any size. By default, this strategy is enabled for all objects. Setting **mru** to "**off**" disables this strategy. If you turn **mru** off for an object, all pages are read into the MRU/LRU chain in cache, and they remain in the cache until they are flushed by additional I/O. For more information on cache strategies, see the *Performance and Tuning Guide*.
- You can change the cache strategy only for objects in the current database.
- When you use **sp_cachestrategy** without specifying the strategy and setting, it reports the current settings for the object, as shown in example 1.
- To see the size, status and I/O size of all data caches on the server, use **sp_cacheconfig**.
- Setting **prefetch "on"** has no effect on tables or indexes that are read into a cache that allows only 2K I/O. The **mru** strategy can be used in all caches, regardless of available I/O size.

Overrides

- If prefetching is turned on for a table or an index, you can override the prefetching for a session with **set prefetch "off"**. If prefetching is turned off for an object, you cannot override that setting.

- The **prefetch**, **lru**, and **mru** options to the **select**, **delete** and **update** commands suggest the I/O size and cache strategy for individual statements. If prefetching or MRU strategy is enabled for a table or an index, you can override it for a query by specifying 2K I/O for **prefetch**, and by specifying **lru** strategy. For example, the following command forces LRU strategy, 2K I/O, and a table scan of the *titles* table:

  ```
  select avg(advance)
  from titles (index titles prefetch 2 lru)
  ```

  If you request a prefetch size, and the object's cache is not configured for I/O of the requested size, the optimizer chooses the best available I/O size.

- If prefetching is enabled for an object with **sp_cachestrategy**, using a **prefetch** specification of 2K in a **select**, **update** or **delete** command overrides an earlier **set prefetch "on"** statement. Specifying a larger I/O size in a **select**, **update** or **delete** command does not override a **set prefetch "off"** command.

  .

Permissions

Only a System Administrator or the object owner can execute **sp_cachestrategy.**

Tables Used

master..sysattributes, *master..sysdatabases*, *sysattributes*, *sysindexes*, *sysobjects*

See Also

Commands	**delete**, **select**, **set**, **update**
Stored procedures	**sp_cacheconfig**, **sp_poolconfig**

sp_changedbowner

Function

Changes the owner of a user database.

Syntax

```
sp_changedbowner loginame [, true ]
```

Parameters

loginame – is the login name of the new owner of the current database.

true – transfers aliases and their permissions to the new database owner. Values are "**true**" and "**TRUE**".

Examples

```
1. sp_changedbowner albert
```

Makes the user "albert" the owner of the current database.

Comments

- The new owner must not already be known as either a user or alias (that is, the new owner must not already be listed in *sysusers* or *sysalternates*). Executing **sp_changedbowner** with the single parameter *loginame* changes the database ownership to *loginame* and drops aliases of users who could act as the old "dbo."
- After executing **sp_changedbowner**, the new owner is known as the Database Owner inside the database.
- **sp_changedbowner** cannot transfer ownership of the system databases.
- The new owner must already have a login name in Adaptive Server, but must **not** have a database user name or alias name in the database. To assign database ownership to such a user, drop the user name or alias entry before executing **sp_changedbowner.**
- To grant permissions to the new owner, a System Administrator must grant them to the Database Owner, since the user is no longer known inside the database under any other name.

Permissions

Only a System Administrator can execute **sp_changedbowner.**

Tables Used

master..syslogins, sysalternates, sysobjects, sysusers

See Also

Commands	**create database**
System procedures	**sp_addlogin**, **sp_dropalias**, **sp_dropuser**, **sp_helpdb**

sp_changegroup

Function

Changes a user's group.

Syntax

sp_changegroup ***grpname, username***

Parameters

grpname – is the name of the group. The group must already exist in the current database. If you use "public" as the *grpname*, enclose it in quotes, because it is a keyword.

username – is the name of the user to be added to the group. The user must already exist in the current database.

Examples

1. **sp_changegroup fort_mudge, albert**

   The user "albert" is now a member of the "fort_mudge" group. It doesn't matter what group "albert" belonged to before.

2. **sp_changegroup "public", albert**

   Removes "albert" from the group he belonged to without making him a member of a new group (all users are always members of "public".)

Comments

- Executing **sp_changegroup** adds the specified user to the specified group. The user is dropped from the group he or she previously belonged to and is added to the one specified by *grpname*.
- New database users can be added to groups at the same time they are given access to the database with **sp_adduser.**
- Groups are used as a collective name for granting and revoking privileges. Every user is always a member of the default group, "public", and can belong to only one other group.
- To remove someone from a group without making that user a member of a new group, use **sp_changegroup** to change the user's group to "public", as shown above in example 2.

- When a user changes from one group to another, the user loses all permissions that he or she had as a result of belonging to the old group and gains the permissions granted to the new group.

Permissions

Only the Database Owner, a System Administrator, or a System Security Officer can execute **sp_changegroup**.

Tables Used

master..syssrvroles, *syscolumns*, *sysobjects*, *sysprotects*, *sysusers*

See Also

Commands	**grant**, **revoke**
System procedures	**sp_addgroup**, **sp_adduser**, **sp_dropgroup**, **sp_helpgroup**

sp_checknames

Function

Checks the current database for names that contain characters not in the 7-bit ASCII set.

Syntax

sp_checknames

Parameters

None.

Examples

1. sp_checknames

```
Looking for non 7-bit ASCII characters in the system tables
of database:
"master"


============================================================
Table.Column name: "syslogins.password"

The following logins have passwords that contain non 7-bit
ASCII characters.  If you wish to change them use "sp_password";
Remember, only the sa and the login itself may examine or change
the syslogins.password column:

 suid   name
 ------ ------------------------------
      1 sa
      2 probe
      3 bogususer
```

Comments

- **sp_checknames** examines the names of all objects, columns, indexes, user names, group names, and other elements in the current database for characters outside of the 7-bit ASCII set. It reports illegal names and gives instructions to make them compatible with the 7-bit ASCII set.
- Run **sp_checknames** in every database on your server after upgrading from a SQL Server of release 4.0.x or 4.2.x, and after using a default character set that was not 7-bit ASCII.

- Follow the instructions in the **sp_checknames** report to correct all non-ASCII names.

Permissions

Any user can execute **sp_checknames**.

Tables Used

sp_checknames uses the following tables when it is executed in any database:

dbo.syscolumns, *dbo.sysindexes*, *dbo.sysobjects*, *dbo.syssegments*, *dbo.systypes*, *dbo.sysusers*

sp_checknames uses the following tables when it is executed in the *master* database:

master..sysdatabases, *master..sysdevices*, *master..syslogins*, *master..sysremotelogins*, *master..sysservers*

See Also

Commands	**update**
System procedures	**sp_password**, **sp_rename**, **sp_renamedb**

sp_checkreswords

Function

Detects and displays identifiers that are Transact-SQL reserved words. Checks server names, device names, database names, segment names, user-defined datatypes, object names, column names, user names, login names, and remote login names.

Syntax

sp_checkreswords [*user_name_param*]

Parameters

user_name_param – is the name of a user in the current database. If you supply *user_name_param*, **sp_checkreswords** checks only for objects that are owned by the specified user.

Examples

1. sp_checkreswords *(executed in master database)*

```
Reserved Words Used as Database Object Names for Database master

Upgrade renames sysobjects.schema to sysobjects.schemacnt.

Owner
------------------------------
dbo

Table                          Reserved Word Column Names
------------------------------ ------------------------------
authorization                  cascade

Object Type                    Reserved Word Object Names
------------------------------ ------------------------------
rule                           constraint
stored procedure               check
user table                     arith_overflow
user table                     authorization

------------------------------------------------------------
------------------------------------------------------------

Owner
------------------------------
lemur
```

```
Table                          Reserved Word Column Names
------------------------------ ------------------------------
key                            close

Table                          Reserved Word Index Names
------------------------------ ------------------------------
key                            isolation

Object Type                    Reserved Word Object Names
------------------------------ ------------------------------
default                        isolation
rule                           level
stored procedure               mirror
user table                     key

Reserved Word Datatype Names
------------------------------
identity


------------------------------------------------------------
------------------------------------------------------------

Database-wide Objects
---------------------


Reserved Word User Names
------------------------------
at
identity

Reserved Word Login Names
------------------------------
at
identity

Reserved Word as Database Names
------------------------------
work

Reserved Word as Language Names
------------------------------
national
```

```
 Reserved Word as Server Names
 -----------------------------
 mirror
 primary

 Reserved Word ServerNetNames
 --------------------------------
 mirror
 primary
```

2. sp_checkreswords *(executed in user database)*

```
Reserved Words Used as Database Object Names for Database user_db

 Upgrade renames sysobjects schema to sysobjects.schemacnt.

 Owner
 ------------------------------
 tamarin

 Table                          Reserved Word Column Names
 ------------------------------ ------------------------------
 cursor                         current
 endtran                        current
 key                            identity
 key                            varying
 schema                         primary
 schema                         references
 schema                         role
 schema                         some
 schema                         user
 schema                         work

 Table                          Reserved Word Index Names
 ------------------------------ ------------------------------
 key                            double

 Object Type                    Reserved Word Object Names
 ------------------------------ ------------------------------
 default                        escape
 rule                           fetch
 stored procedure               foreign
 user table                     cursor
 user table                     key
 user table                     schema
```

```
view                            endtran


------------------------------------------------------------
------------------------------------------------------------

Database-wide Objects
---------------------

Found no reserved words used as names for database-wide objects.
```

Comments

- **sp_checkreswords** reports the names of existing objects that are reserved words. Transact-SQL does not allow words that are part of any command syntax to be used as identifiers, unless you are using delimited identifiers. Reserved words are pieces of SQL syntax, and they have special meaning when you use them as part of a command. For example, in pre-release 10.0 SQL Server, you could have a table called *work*, and select data from it with this query:

  ```
  select * from work
  ```

  work was a new reserved word in SQL Server release 10.0, part of the command **commit work**. Issuing the same **select** statement in release 10.0 or later causes a syntax error. **sp_checkreswords** finds identifiers that would cause these problems.

- **sp_checkreswords** also finds reserved words, used as identifiers, that were created using the **set quoted_identifier** option.

- Use **sp_checkreswords** before or immediately after upgrading to a new release of Adaptive Server. For information on installing and running this procedure before performing the upgrade, see the installation documentation for your platform.

  Run **sp_checkreswords** in the *master* database and in each user database. Also run it in *model* and *sybsystemprocs*, if you have added users or objects to those databases.

- The return status indicates the number of items found.

- If you supply a user name, **sp_checkreswords** checks for all of the objects that can be owned by a user tables, indexes, views, procedures, triggers, rules, defaults, and user-defined datatypes. It reports all identifiers that are reserved words.

- If your current database is not the *master* database, and you do not provide a user name, **sp_checkreswords** checks for all of the objects above, with a separate section in the report for each user

name. It also checks *sysusers* and *syssegments* for user names and segment names that are reserved words. You only need to check *model* and *sybsystemprocs* if you have added objects, users, or user-defined datatypes.

- If your current database is *master*, and you do not provide a user name, **sp_checkreswords** performs all of the checks above and also checks *sysdatabases*, *syslogins*, *syscharsets*, *sysservers*, *sysremotelogins*, *sysdevices*, and *syslanguages* for reserved words used as the names of databases, local or remote logins, local and remote servers, character sets, and languages.

Handling Reported Instances of Reserved Words

- If **sp_checkreswords** reports that reserved words are used as identifiers, you have two options:
  - Use **sp_rename**, **sp_renamedb**, or update the system tables to change the name of the identifier.
  - Use **set quoted_identifier on** if the reserved word is a table name, view name, or column name. If most of your applications use stored procedures, you can drop and re-create these procedures with **set quoted_identifier on**, and quote all identifiers. All users will be able to run the procedures, without having to use **set quoted_identifier on** for their session. You can use **set quoted_identifier on**, create views that give alternative names to tables or columns, and change your applications to reference the view instead.

    The following example provides alternatives for the new reserved words “key”, “level”, and “work”:

    ```
    create view keyview
    as
    select lvl = "level", wrk = "work"
    from "key"
    ```

    The syntax for the **set** command is:

    ```
    set quoted_identifier on
    ```

- If you do not either change the identifiers or use delimited identifiers, any query that uses the reserved words as identifiers reports an error, usually a syntax error. For example:

  ```
  select level, work from key
  ```

  ```
  Msg 156, Level 15, State 1:
  Server 'rosie', Line 1:
  Incorrect syntax near the keyword 'level'.
  ```

➤ ***Note***

The quoted identifier option is a SQL92 option and may not be supported by many client products that support other Adaptive Server features. For example, you cannot use **bcp** on tables whose names are reserved words. Before choosing the quoted identifier option, perform a test on various objects using all the tools you will use to access Adaptive Server. Use **set quoted_identifier on**, create a table with a reserved word for a name and reserved words for column names. If the client product generates SQL code, it must enclose identifiers in double quotes (if they are reserved words) and character constants in single quotes.

- Procedures, triggers, and views that depend on objects whose names have been changed may work after the name change, but will stop working when the query plan is recompiled. Recompilation takes place for many reasons, without notification to the user. To avoid unsuspected loss of functionality, change the names of objects in procedures, triggers, and views immediately after you change the object name.
- Whether you change the object names or use delimited identifiers, you must change all stored procedures, views, triggers, and applications that include the reserved word. If you change object names, you must change identifiers; if you use delimited identifiers, you must add the **set quoted_identifier** option and quotation marks.
- If you do not have the text of your procedures, triggers, views, rules, and defaults saved in operating system files, you can use **defncopy** to copy the definitions from the server to files. See **defncopy** in the *Utility Programs* manual for your platform.

Changing Identifiers

- If you change the names of the items reported by **sp_checkreswords**, you must change the names in all procedures, triggers, views, and applications that reference the object using the reserved word.
- Dump your database before changing identifier names. After you change the identifier names, run **dbcc** to determine that there are no problems, and dump the database again.

- If you are changing identifiers on an active production database:
  - Perform the changes when the system is least busy, so that you will disrupt as few users as possible.
  - Prepare carefully by finding all Open Client DB-Library™ programs, windowing applications, stored procedures, triggers, and scripts that use a particular identifier. This way, you can make the edits needed in the source code, then change the identifiers and replace the procedures and code as quickly as possible.
- The procedure **sp_depends** can help find procedures, views, and triggers that use table and view names.

Using *sp_rename* to Change Identifiers

- The system procedure **sp_rename** renames tables, indexes, views, procedures, triggers, rule, defaults, user-defined datatypes, and columns. Use **sp_renamedb** to rename databases.
- Table 7-10 shows the types of identifiers that you can change with **sp_rename** and lists other changes that may have to be made on the server and in your application programs.

Table 7-10: sp_rename and changing identifiers

Identifier	Remember To
Table name	• Drop all procedures, triggers and views that reference the table, and re-create them with the new name. Use **sp_depends** to find the objects that depend on the table. • Change all applications or SQL source scripts that reference the table to use the new table name. • Change **dbcc** scripts that perform table-level checks using table names.
Index name	• Drop any stored procedures that create or drop the index, and re-create them with the new name. • Change all applications or SQL source scripts that create or drop the index. • Change **dbcc** scripts that perform index-level checks using index names.

Table 7-10: sp_rename and changing identifiers (continued)

Identifier	Remember To
View name	• Drop all procedures, triggers, and views that reference the view, and re-create them with the new name. Use **sp_depends** to find the objects that depend on the view. • Change all applications or SQL source scripts that reference the view to use the new view name.
Procedure name	• Drop and re-create with the new procedure name all procedures and triggers that reference the procedure. • Change all applications or SQL source scripts that execute the procedure to use the new name. • If another server remotely calls the procedure, change applications on the remote server to use the new procedure name.
Trigger name	• Change any SQL source scripts that create the trigger.
Rule name	• Change any SQL source scripts that create the rule.
Default name	• Change any SQL source scripts that create the default.
User-defined datatype name	• Drop all procedures that create tables with user-defined datatypes, and re-create them with the new name. • Change any applications that create tables with user-defined datatypes.

Column name	• Drop all procedures, triggers and views that reference the column, and re-create them with the new column name.
	• **sp_depends** cannot find column name references. The following query displays the names of procedures, triggers, and views that reference a column named "key":
	`select distinct sysobjects.name`
	`from sysobjects, syscomments`
	`where sysobjects.id = syscomments.id`
	`and syscomments.text like "%key%"`
	• Change all applications and SQL source scripts that reference the column by name.

The following command changes the name of the view *isolation* to *isolated*:

```
sp_rename "isolation", isolated
```

The following command changes the name of a column in the renamed view *isolated*:

```
sp_rename "isolated.key", keyname
```

- Use **sp_depends** to get a list of all views, procedures, and triggers that reference a view, procedure, or table that will be renamed. To use **sp_depends** after renaming an object, give the new name. For example:

  sp_depends ***new_name***

Renaming Databases with *sp_renamedb*

- To change the name of a database, use **sp_renamedb**. The database must be in single-user mode. Drop and re-create any procedures, triggers, and views that explicitly reference the database name. For more information, see **sp_renamedb**.

Changing Other Identifiers

- To change user names, login names, device names, remote server names, remote server user names, segment names, and character set and language names, first determine if you can drop the object or user, then add or create it again. If you cannot do that, use the command:

  sp_configure "allow updates to system tables", 1

  to allow direct updates to system tables. Only a System Security Officer can set the **allow updates to system tables** configuration parameter.

  Errors during direct updates to system tables can create severe problems in Adaptive Server. To determine whether you can drop the objects or user, then re-create them, see Table 7-11.

  Table 7-13: Considerations when changing identifiers on page 1-129 shows possible dependencies on this set of identifiers. See this table for possible dependencies, whether you choose to upgrade by dropping and recreating objects, by using delimited identifiers, or by performing direct updates to system tables.

Table 7-11: Alternatives to direct system tables updates when changing identifiers

Identifier Type	Suggested Actions to Avoid Updates to System Tables
User names and login names	To change the name of a user with no objects, first use **sp_helprotect** *username* in each database to record the user's permissions. Then, drop the user from all of the databases (**sp_dropuser**), and drop the login (**sp_droplogin**). Finally, add the new login name (**sp_addlogin**), add the new user name to the databases (**sp_adduser**), and restore the user's permissions with **grant**.

Table 7-11: Alternatives to direct system tables updates when changing identifiers (continued)

Identifier Type	Suggested Actions to Avoid Updates to System Tables
Device names	If this device is completely allocated, you will not need to use its name in a **create database** command, so you can leave the name unchanged.
Remote server names	Unless there are large numbers of remote login names from the remote server, drop the remote server (**sp_dropserver**) and add it with a new name (**sp_addserver**).
Remote server logins	Drop the remote login with **sp_dropremotelogin**, add it with a new name using **sp_addremotelogin**, and restore the user's permission to execute procedures with **grant**.
Segment names	These are rarely used, once objects have been created on the segments.
Character set and language names	Languages and character sets have reserved words as identifiers only if a System Administrator has created alternative languages with **sp_addlanguage**. Drop the language with **sp_droplanguage**, and add it with a new name.

◆ ***WARNING!***

Direct updates to system tables can be very dangerous. You can make mistakes that make it impossible for Adaptive Server to run or make it impossible to access objects in your databases. Undertake this effort when you are calm and collected, and when little or no production activity is taking place on the server. If possible, use the alternative methods described Table 7-11.

- The following example shows a "safe" procedure for updating a user name, with all data modification preceded by a **begin transaction** command:

  The System Security Officer executes the following command:

  ```
  sp_configure "allow updates to system tables", 1
  ```

  Then you can execute the following:

  ```
  begin transaction
  update sysusers
  set name = "workerbee"
  where name = "work"
  ```

  At this point, run the query, and check to be sure that the command affected only the row that you intended to change. The only identifier change that affects more than one row is changing the *language* name in *syslogins*.

- If the query affected only the correct row, use **commit transaction**.
- If the query affected more than one row, or the incorrect row, use **rollback transaction**, determine the source of the problem, and execute the command correctly.

When you are finished, the System Security Officer turns off the **allow updates to system tables** configuration parameter with this command:

```
sp_configure "allow updates to system tables", 0
```

◆ ***WARNING!***

Only update system tables in a single database in each user defined transaction. Do not issue a begin transaction command and then update tables in several databases. Such actions can make recovery extremely difficult.

Table 7-12 shows the system tables and columns that you should update to change reserved words. The tables preceded by "*master.dbo.*" occur only in the *master* database. All other tables occur in *master* and in user databases. Be certain you are using the correct database before you attempt the update. You can check for the current database name with this command:

```
select db_name()
```

Table 7-12: System table columns to update when changing identifiers

Type of Identifier	Table to Update	Column Name
User name	*sysusers*	*name*
Login names	*master.dbo.syslogins*	*name*
Segment names	*syssegments*	*name*
Device name	*sysdevices*	*name*
Remote server name	*sysservers*	*srvname*
Remote server network name	*sysservers*	*srvnetname*
Character set names	*master.dbo.syscharsets*	*name*
Language name	*master.dbo.syslanguages* *master.dbo.syslogins*	*name* *language*

Table 7-13 shows other changes that may have to be made on the server and in your application programs:

Table 7-13: Considerations when changing identifiers

Identifier	Remember To
Login name	Change the user name in each database where this person is a user.
User name	Drop, edit, and re-create all procedures, triggers, and views that use qualified (*owner_name.object_name*) references to objects owned by this user. Change all applications and SQL source scripts that use qualified object names to use the new user name. You do not have to drop the objects themselves; *sysusers* is linked to *sysobjects* by the column that stores the user's ID, not the user's name.
Device name	Change any SQL source scripts or applications that reference the device name to use the new name.
Remote server name	Change the name on the remote server. If the name that **sp_checkreswords** reports is the name of the local server, you must restart the server before you can issue or receive remote procedure calls.
Remote server network name	Change the server's name in the interfaces files.
Remote server login name	Change the name on the remote server.
Segment name	Drop and re-create all procedures that create tables or indexes on the segment name. Change all applications that create objects on segments to use the new segment name.
Character set name	None.
Language name	Change both *master.dbo.syslanguages* and *master.dbo.syslogins*. The update to *syslogins* may involve many rows. Also, change the names of your localization files.

Using Delimited Identifiers

- You can use delimited identifiers for table names, column names, and view names. You cannot use delimited identifiers for other object names.
- If you choose to use delimited identifiers, use **set quoted_identifier on**, and drop and re-create all the procedures, triggers, and views that use the identifier. Edit the text for those objects, enclosing the

reserved words in double quotes and enclosing all character strings in single quotes.

The following example shows the changes to make to queries in order to use delimited identifiers. This example updates a table named *work*, with columns named *key* and *level*. Here is the pre-release 10.0 query, which encloses character literals in double quotes, and the edited version of the query for use with delimited identifiers:

```
/* pre-release 10.0 version of query */
update work set level = "novice"
    where key = "19-732"

/* 10.0 or later version of query, using
** the quoted identifier option
*/
update "work" set "level" = 'novice'
    where "key" = '19-732'
```

- All applications that use the reserved word as an identifier must be changed as follows:
  - The application must set the quoted identifier option on.
  - All uses of the reserved word must be enclosed in double quotes.
  - All character literals used by the application while the quoted identifier option is turned on must be enclosed in single quotes. Otherwise, Adaptive Server attempts to interpret them as object names.

  For example, the following query results in an error message:

  ```
  set quoted_identifier on

  select * from titles where title_id like "BU%"
  ```

  Here is the correct query:

  ```
  select * from titles where title_id like 'BU%'
  ```

- Stored procedures that you create while the delimited identifiers are in effect can be run without turning on the option. (The **allow updates to system tables** option also works this way.) This means that you can turn on quoted identifier mode, drop a stored procedure, edit it to insert quotation marks around reserved words used as identifiers, and re-create the procedure. All users can execute the procedure without using **set quoted_identifier.**

Permissions

Only a System Administrator can execute **sp_checkreswords**.

Tables Used

master..syscharsets, *master..sysdatabases*, *master..sysdevices*, *master..syslanguages*, *master..syslogins*, *master..sysremotelogins*, *master..sysservers*, *master..sysmessages*, *syscolumns*, *sysindexes*, *sysobjects*, *syssegments*, *systypes*, *sysusers*

See Also

Commands	**set**
System procedures	**sp_configure**, **sp_depends**, **sp_rename**, **sp_renamedb**
Utility commands	**defncopy**

sp_checksource

Function

Checks for the existence of the **source text** of the **compiled object**.

Syntax

sp_checksource [*objname* [, *tabname* [, *username*]]]

Parameters

objname – is the compiled object to be checked for the existence of its source text.

tabname – is the name of the table or view to be checked for the existence of all check constraints, defaults, and triggers defined on it.

username – is the name of the user who owns the compiled objects to be checked for the existence of the source text.

Examples

1. **sp_checksource**

   Checks for the existence of the source text of all compiled objects in the current database.

2. **sp_checksource titleview**

   Checks for the existence of the source text of the view named *titleview.*

3. **sp_checksource title_vu, @username = Mary**

   Checks for the existence of the source text of the view named *titls_vu* that is owned by Mary.

4. **sp_checksource list_phone_proc**

   Checks for the existence of the source text of the custom stored procedure *list_phone_proc.*

5. **sp_checksource @tabname = "my_tab"**

   Checks for the existence of the source text of all the check constraints, triggers, and declarative defaults defined on the table named *my_tab.*

6. **`sp_checksource @objname = "my_vu", @tabname = "my_tab"`**

   Checks for the existence of the source text of the view *my_vu* and all check constraints, triggers, and defaults defined on the table *my_tab*.

7. **`sp_checksource @username = "Tom"`**

   Checks for the existence of the source text of all compiled objects owned by Tom.

Comments

- **sp_checksource** checks for the existence of the source text of the specified compiled object. If the source text exists for the specified object, **sp_checksource** returns 0. If the source text does not exist for the specified object, **sp_checksource** returns 1.
- If you do not provide any parameters, **sp_checksource** checks the existence of the source text for all compiled objects in the current database.
- To use **sp_checksource** with no parameters, you must be the Database Owner or System Administrator.

Permissions

Only a Database Owner or System Administrator can execute **sp_checksource** to check for the existence of the source text of compiled objects that are owned by another user. Any user can execute **sp_checksource** to check for the existence of the source text for his or her own compiled objects.

Tables Used

syscolumns, *syscomments*, *sysconstraints*, *sysobjects*, *sysprocedures*

See Also

System procedures	**sp_hidetext**

sp_chgattribute

Function

Changes the **max_rows_per_page**, **fillfactor**, **reservepagegap**, or **exp_row_size** value for future space allocations of a table or an index; sets the **concurrency_opt_threshold** for a table.

Syntax

```
sp_chgattribute objname, {"max_rows_per_page" |
   "fillfactor" | "reservepagegap" | "exp_row_size"
   concurrency_opt_threshold }, optvalue
```

Parameters

objname – is the name of the table or index for which you want to change attributes.

max_rows_per_page – specifies the row size. Use this option for tables with variable-length columns.

fillfactor – specifies how full Adaptive Server will make each page when it is re-creating an index or copying table pages as a result of a **reorg rebuild** command or an **alter table** command to change the locking scheme. The **fillfactor** percentage is relevant only at the time the index is rebuilt. Valid values are 0–100.

reservepagegap – specifies the ratio of filled pages to empty pages that are to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

exp_row_size – reserves a specified amount of space for the rows in data-only locked tables. Use this option to reduce the number of rows being forwarded, which can be expensive during updates. Valid values are 0, 1, and any value between the minimum and maximum row length for the table. 0 means a server-wide setting is applied, and 1 means to fully pack the rows on the data pages.

concurrency_opt_threshold – specifies the table size, in pages, at which access to a data-only-locked table should begin optimizing for reducing I/O, rather than for concurrency. If the table is smaller than the number of pages specified by **concurrency_opt_threshold**, the query is optimized for concurrency by always using available indexes; if the table is larger than the number of pages specified by **concurrency_opt_threshold**, the query is optimized for I/O instead.

Valid values are -1 to 32767. Setting the value to 0 disables concurrency optimization. Use -1 to enforce concurrency optimization for tables larger than 32767 pages. The default is 15 pages.

optvalue – is the new value. Valid values and default values depend on which parameter is specified.

Examples

1. `sp_chgattribute authors, "max_rows_per_page", 1`

   Sets the **max_rows_per_page** to 1 for the *authors* table for all future space allocations.

2. `sp_chgattribute "titles.titleidind", "max_rows_per_page", 4`

   Sets the **max_rows_per_page** to 4 for the *titleidind* index for all future space allocations.

3. `sp_chgattribute "titles.title_ix", "fillfactor", 90`

   Specifies a **fillfactor** of 90 percent for pages in *title_ix*.

4. `sp_chgattribute authors, "exp_row_size", 120`

   Sets the **exp_row_size** to 120 for the *authors* table for all future space allocations.

5. `sp_chgattribute "titles.titleidind", "reservepagegap", 16`

   Sets the **reservepagegap** to 16 for the *titleidind* index for all future space allocations.

6. `sp_chgattribute "titles", concurrency_opt_threshold, 0`

   Turns off concurrency optimization for the *titles* table.

Comments

- **sp_chgattribute** changes the **max_rows_per_page, fillfactor, reservepagegap,** or **exp_row_size** value for future space allocations or data modifications of the table or index. It does not affect the space allocations of existing data pages. You can change these values for an object only in the current database.
- Setting **max_rows_per_page** to 0 tells Adaptive Server to fill the data or index pages and not to limit the number of rows (this is the default behavior of Adaptive Server if **max_rows_per_page** is not set).

- Low values for *optvalue* may cause page splits. Page splits occur when new data or index rows need to be added to a page, and there is not enough room for the new row. Usually, the data on the existing page is split fairly evenly between the newly allocated page and the existing page.
- To approximate the maximum value for a nonclustered index, subtract 32 from the page size and divide the resulting number by the index key size. The following statement calculates the maximum value of **max_rows_per_page** for the nonclustered index *titleind*:

```
select
    (select @@pagesize - 32) / minlen
     from sysindexes where name = "titleind"

-----------
        288
```

  If you specify too high a value for *optvalue*, Adaptive Server returns an error message specifying the highest value allowed.
- If you specify an incorrect value for **max_rows_per_page**, **fillfactor**, **reservepagegap**, or **exp_row_size**, **sp_chgattribute** returns an error message specifying the valid values.
- For more information on **max_rows_per_page**, **fillfactor**, **reservepagegap**, **exp_row_size**, and **concurrency_opt_threshold**, see the *Performance and Tuning Guide*.

Permissions

Only the object owner can execute **sp_chgattribute**.

Tables Used

sysindexes, *sysobjects*, *systabstats*

See Also

Commands	**alter table**, **create index**, **create table**
System procedures	**sp_helpindex**

sp_clearpsexe

Function

Clears the execution attributes of an Adaptive Server session that was set by **sp_setpsexe**.

Syntax

```
sp_clearpsexe spid, exeattr
```

Parameters

spid – is the process ID of the session for which execution attributes are to be cleared.

exeattr – identifies the execution attributes to be cleared. Values for *exeattr* are "**priority**" and **"enginegroup"**.

Examples

```
1. sp_clearpsexe 12, 'enginegroup'
```

Drops the engine group entry for process 12.

Comments

- **sp_clearpsexe** clears the execution attributes of the session that was set by **sp_setpsexe**. For more information, see the *Performance and Tuning Guide*.
- If the execution attributes are not cleared during the lifetime of the session, they are cleared when the session exits or terminates abnormally.
- **sp_clearpsexe** fails if there are no online engines in the associated engine group.
- When you drop an engine group entry, the session executes on an engine group determined by a class definition or by the default class.
- Use **sp_who** to list process IDs (*spid*s).

Permissions

Only a System Administrator can execute **sp_clearpsexe** to clear priority attributes for all users. Any user can execute **sp_clearpsexe** to clear the priority attributes of tasks owned by that user.

Tables Used

sysattributes, sysprocesses

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_dropengine, sp_dropexeclass, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**

sp_clearstats

Function

Initiates a new accounting period for all server users or for a specified user. Prints statistics for the previous period by executing **sp_reportstats**.

Syntax

sp_clearstats [*loginame*]

Parameters

loginame – is the user's login name.

Examples

1. sp_clearstats

```
Name    Since           CPU  Percent CPU     I/O    Percent I/O
------  --------     ------  ----------- -------    -------------
probe   Jun 19 1990       0           0%       0         0%
julie   Jun 19 1990   10000     24.9962%    5000         24.325%
jason   Jun 19 1990   10002     25.0013%    5321         25.8866%
ken     Jun 19 1990   10001     24.9987%    5123         24.9234%
kathy   Jun 19 1990   10003     25.0038%    5111         24.865%
(5 rows affected)
             Total CPU   Total I/O
             ---------      ---------
             40006       20555
             5 login accounts cleared.
```

Initiates a new accounting period for all users.

2. sp_clearstats kathy

```
Name  Since         CPU     Percent CPU     I/O     Percent I/O
----- -----------   -----   ------------    -----   -----------
KATHY Jul 24 1990   498     49.8998%        483924  9.1829%
(1 row affected)
             Total CPU   Total I/O
             ---------   ----------
             998         98392
             1 login account cleared.
```

Initiates a new accounting period for the user "kathy."

Comments

- **sp_clearstats** creates an accounting period and should be run only at the end of a period.
- Because **sp_clearstats** clears out the accounting statistics, you must record the statistics **before** running the procedure.
- **sp_clearstats** updates the *syslogins* field *accdate* and clears the *syslogins* fields *totcpu* and *totio*.

Permissions

Only a System Administrator can execute **sp_clearstats**.

Tables Used

master..syslogins, sysobjects

See Also

System procedures	**sp_reportstats**

sp_cmp_all_qplans

Function

Compares all abstract plans in two abstract plan groups.

Syntax

sp_cmp_all_qplans *group1, group2* [, *mode*]

Parameters

group1, *group2* – are the names of the 2 abstract plan groups.

mode – is the display option, one of: **counts**, **brief**, **same**, **diff**, **first**, **second**, **offending** and **full**. The default mode is **counts**.

Examples

1. sp_cmp_all_qplans dev_plans, prod_plans

```
If the two query plans groups are large, this
might take some time.
Query plans that are the same
 count
 -----------
          49
Different query plans that have the same
association key
 count
 -----------
           1
Query plans present only in group 'dev_plans' :
 count
 -----------
           1
Query plans present only in group 'prod_plans' :
 count
 -----------
           0
```

Generate a default report on 2 abstract plan groups.

2. sp_cmp_all_qplans **dev_plans, prod_plans**, brief

Generates a report using the **brief** mode.

Comments

- Use **sp_cmp_all_qplans** to check for differences in abstract plans in two groups of plans.
- **sp_cmp_all_qplans** matches pairs of plans where the plans in each group have the same user ID and query text. The plans are classified as follows:
  - Plans that are the same
  - Plans that have the same association key in both groups, but have different abstract plans. The association key is the group ID, user ID and query text.
  - Plans that exist in one group, but do not exist in the other group

  Table 7-14 shows the report modes and what type of information is reported for each mode.

Table 7-14: Report modes for sp_cmp_all_qplans

Mode	Reported Information
counts	The counts of: plans that are the same, plans that have the same association key, but different groups, and plans that exist in one group, but not the other. This is the default report mode.
brief	The information provided by counts, plus the IDs of the abstract plans in each group where the plans are different, but the association key is the same, and the IDs of plans that are in one group, but not in the other.
same	All counts, plus the IDs, queries, and plans for all abstract plans where the queries and plans match.
diff	All counts, plus the IDs, queries, and plans for all abstract plans where the queries and plans are different.
first	All counts, plus the IDs, queries, and plans for all abstract plans that are in the first plan group, but not in the second plan group.
second	All counts, plus the IDs, queries, and plans for all abstract plans that are in the second plan group, but not in the first plan group.
offending	All counts, plus the IDs, queries, and plans for all abstract plans that have different association keys or that do not exist in both groups. This is the combination of the diff, first and second modes
full	All counts, plus the IDs, queries, and plans for all abstract plans. This is the combination of same and offending modes.

- To compare two individual abstract plans, use **sp_cmp_qplans**. To see the names of abstract plan groups, use **sp_help_qpgroup**.
- When a System Administrator or Database Owner runs **sp_cmp_all_qplans**, it reports on all plans in the two groups. When another user executes **sp_cmp_all_qplans**, it reports only on plans that have the user's ID.

Permissions

Any user can execute **sp_cmp_all_qplans**.

Tables Used

sysattributes, sysqueryplans

See Also

System procedures	**sp_cmp_qplans**

sp_cmp_qplans

Function

Compares two abstract plans.

Syntax

sp_cmp_qplans ***id1, id2***

Parameters

id1, *id2* – are the IDs of two abstract plans.

Examples

1. sp_cmp_qplans 411252620, 1383780087

```
The queries are the same.
The query plans are the same.
```

2. sp_cmp_qplans 2091258605, 647777465

```
The queries are the same.
The query plans are different.
```

Comments

- **sp_cmp_qplans** compares the queries, abstract plans, and hash keys of two abstract plans, and reports whether the queries are the same, and whether the plans are the same. It prints one of these messages for the query:
  - The queries are the same.
  - The queries are different.
  - The queries are different but have the same hash key.

  It prints one of these messages for the abstract plan:
  - The query plans are the same.
  - The query plans are different.

- **sp_cmp_qplans** also prints a return status showing the results of the comparison. The status values 1, 2 and 10 are additive. The status values are show in Table 7-15

Table 7-15: Return status values for sp_cmp_qplans

Return value	Meaning
0	The query text and abstract plans are the same.
+1	The queries and hash keys are different.
+2	The queries are different, but the hash keys are the same.
+10	The abstract plans are different.
100	One or both of the plan IDs does not exist.

- To find the ID of a plan, use **sp_help_qpgroup** or **sp_find_qplan**. Plan IDs are also returned by **create plan** and are included in **showplan** output.

Permissions

Any user can execute **sp_cmp_qplans** to compare plans that he or she owns. Only a System Administrator or the Database Owner can compare plans owned by another user.

Tables Used

sysqueryplans

See Also

System procedures	**sp_cmp_all_qplans**, **sp_help_qpgroup**

sp_commonkey

Function

Defines a common key—columns that are frequently joined—between two tables or views.

Syntax

```
sp_commonkey tabaname, tabbname, col1a, col1b
     [, col2a, col2b, ..., col8a, col8b]
```

Parameters

tabaname – is the name of the first table or view to be joined.

tabbname – is the name of the second table or view to be joined.

col1a – is the name of the first column in the table or view *tabaname* that makes up the common key. Specify at least one pair of columns (one column from the first table or view and one from the second table or view).

col1b – is the name of the partner column in the table or view *tabbname* that is joined with *col1a* in the table or view *tabaname*.

Examples

1. **`sp_commonkey titles, titleauthor, title_id, title_id`**

   Defines a common key on *titles.titleid* and *titleauthor.titleid*.

2. **`sp_commonkey projects, departments, empid, empid`**

   Assumes two tables, *projects* and *departments*, each with a column named *empid*. This statement defines a frequently used join on the two columns.

Comments

- Common keys are created in order to make explicit a logical relationship that is implicit in your database design. The information can be used by an application. **sp_commonkey** does not enforce referential integrity constraints; use the **primary key** and **foreign key** clauses of the **create table** or **alter table** command to enforce key relationships.

- Executing **sp_commonkey** adds the key to the *syskeys* system table. To display a report on the common keys that have been defined, use **sp_helpkey.**
- You must be the owner of at least one of the two tables or views in order to define a common key between them.
- The number of columns from the first table or view must be the same as the number of columns from the second table or view. Up to eight columns from each table or view can participate in the common key. The datatypes of the common columns must also agree. For columns that take a length specification, the lengths can differ. The null types of the common columns need not agree.
- The installation process runs **sp_commonkey** on appropriate columns of the system tables.

Permissions

Only the owner of *tabaname* or *tabbname* can execute **sp_commonkey.**

Tables Used

syscolumns, *syskeys*, *sysobjects*

See Also

Commands	**alter table**, **create table**, **create trigger**
System procedures	**sp_dropkey**, **sp_foreignkey**, **sp_helpjoins**, **sp_helpkey**, **sp_primarykey**

sp_companion

Function

Performs cluster operations such as configuring Adaptive Server as a secondary companion in a high availability system and moving a companion server from one failover mode to another. **sp_companion** is run from the secondary companion.

Syntax

```
sp_companion
   [server_name
   {,configure
       [,{with_proxydb | NULL}]
       [,srvlogin]
       [,server_password]
       [,cluster_login]
       [,cluspassword]]
   | drop
   | suspend
   | resume
   | prepare_failback
   | do_advisory}
          {, all
          | help
          | group attribute_name
          | base attribute_name}
```

Parameters

server_name – Is the name of the Adaptive Server on which you are performing a cluster operation.

configure – Configures the server specified by *server_name* as the primary companion in a failover configuration.

drop – Permanently drops a companion from failover configuration. After the command has completed, the servers are in single-server mode.

suspend – Temporarily removes the companions from a failover configuration. After the command is completed, the companions are in suspended mode.

resume – Reverses the **suspend** command and resumes normal companion mode between the companions.

prepare_failback – Prepare the secondary companion to relinquish the primary companion's resources so it can failback.

do_advisory – Verifies that the secondary companion is compatible for successfully performing the primary companion's functions during failover mode.

all – Causes **do_advisory** the investigate all the parameters.

help – Displays information and syntax about the **do_advisory** parameter.

group attribute – Limits **do_advisory** to investigate only the group attributes.

base attribute – Limits **do_advisory** to investigate only the base attributes.

with_proxydb – Creates proxy databases on the secondary companion for all database other than the system databases – and all subsequent databases that are added – when this parameter is included in the initial configuration of the companion servers. By default, **with_proxydb** is disabled.

srvlogin – Is a user's login to access the companion server. By default, the value of **srvlogin** is "sa".

srvpassword – Is the user's password to access the companion server. By default, the value of **srvpassword** is null.

cluster_login –Is the user's login to log into the cluster. By default, the value of **cluster_login** is sa.

cluspassword – Is the users password you must provide to log into the cluster. By default, the value of **cluspassword** is null.

Examples

1. **`sp_companion "MONEY1", configure`**

   Configures the Adaptive Server MONEY1 as the primary companion.

2. **`sp_companion "MONEY1", configure, with_proxydb, "sa", "sapsswd"`**

   Configures the Adaptive Server MONEY1 as the primary companion and creates proxy databases on the secondary companion.

3. **`sp_companion "PERSONEL1", "drop"`**

Drops the Adaptive Server PERSONEL1 from the failover configuration. After the command has completed, both the primary companion and the secondary companion will be in single-server mode.

4. **sp_companion "MONEY1", "resume"**

   Resumes normal companion mode for the companion server (in this example, MONEY1).

5. **sp_companion "PERSONEL1", "prepare_failback"**

   Prepares the primary companion (in this case, PERSONEL1) to change to normal companion mode and resume control of the Adaptive Server that failed over.

6. **sp_companion "PERSONEL1", do_advisory, "all"**

   Checks to make sure a cluster operation with the PERSONEL1 companion will be successful. Because **do_advisory** in this example uses the **all** parameter, it checks all the **do_advisory** attributes of PERSONEL1 to make sure that none of them will prevent a successful cluster operation, and makes sure that the secondary companion can successfully perform the primary companion's operations after failover is complete.

7. **sp_companion "PERSONEL1, do_advisory, "CIS"**

   Checks to make sure that none of the attributes for the CIS (Component Integration Services) on the companion server is compatible with the local server.

Comments

- **sp_companion** performs cluster operations such as configuring Adaptive Server as a secondary companion in a high availability system. **sp_companion** also moves companion servers from one failover mode to another (for example, from failover mode back to normal companion mode). **sp_companion** is run from the secondary companion.
- **sp_companion** is installed with the *installhasvss* (*insthasv* on Windows NT), not the *installmaster* script. *installhasvss* is located in *$SYBASE/ASE-12_0/scripts*
- **sp_companion** automatically disables Sybase's mirroring. Sybase recommends that you use a third-party mirroring software to protect your data from disk failures.

For complete information, see *Using Sybase Failover in A High Availability System*. Before running the **do_advisory** command, make

sure to read the configuration chapter of this book as well as the **do_advisory** chapter.

Permissions

Only users with the **ha_role** can issue **sp_companion**.

sp_configure

Function

Displays or changes configuration parameters.

Syntax

```
sp_configure [configname [, configvalue] | group_name
   | non_unique_parameter_fragment]
sp_configure "configuration file", 0, {"write" |
   "read" | "verify" | "restore"} "file_name"
```

Parameters

Syntax	Effect
sp_configure	Displays configuration parameters by group, their current values, their default values, the value to which they have most recently been set, and the amount of memory used by this setting. Displays only the parameters whose display level is the same as or below that of the user.
sp_configure *configname*	Displays the current value, default value, most recently changed value, and amount of memory used by the setting for all parameters matching *parameter*.
sp_configure *configname*, *configvalue*	Resets *configname* to *configvalue* and displays the current value, default value, configured value, and amount of memory used by *configname*.
sp_configure *configname*, 0, "default"	Resets *configname* to its default value and displays current value, default value, configured value, and amount of memory used by *configname*.
sp_configure *group_name*	Displays all configuration parameters in *group_name*, their current values, their default values, the value (if applicable) to which they have most recently been set, and the amount of memory used by this setting.
sp_configure *non_unique_parameter_fragment*	Displays all parameter names that match *non_unique_parameter_fragment*, their current values, default values, configured values, and the amount of memory used.
sp_configure "configuration file", 0, "write", "*file_name***"**	Creates *file_name* from the current configuration. If *file_name* already exists, a message is written to the error log and the existing file is renamed using the convention *file_name.001*, *file_name.002*, and so on. If you have changed a static parameter but have not restarted your server, **"write"** gives you the currently running value for that parameter.

Syntax	Effect
sp_configure "configuration file", 0, "read", "*file_name***"**	Performs validation checking on values contained in *file_name* and reads those values that pass validation into the server. If any parameters are missing from *file_name*, the current running values for those parameters are used.
sp_configure "configuration file", 0, "verify", "*file_name***"**	Performs validation checking on the values in *file_name*.
sp_configure "configuration file", 0, "restore", "*file_name***"**	Creates *file_name* with the values in *sysconfigures*. This is useful if all copies of the configuration file have been lost and you need to generate a new copy.

Examples

1. **sp_configure**

   Displays all configuration parameters by group, their current values, their default values, the value (if applicable) to which they have most recently been set, and the amount of memory used by this setting.

2. **sp_configure "identity"**

```
Configuration option is not unique.

Parameter Name              Default   Memory Used Config Value Run Value
--------------------------- --------- ----------- ------------ ---------
identity burning set factor      5000           0         5000      5000
identity grab size                  1           0            1         1
size of auto identity column       10           0           10        10
```

   Displays all configuration parameters that include the word "identity."

3. **sp_configure "recovery interval in minutes", 3**

```
Parameter Name              Default Memory Used Config Value Run Value
--------------------------- ------- ----------- ------------ -----------
recovery interval in minutes      5           0            3           3
Configuration option changed. The SQL Server need not be rebooted
since the option is dynamic.
```

   Sets the system **recovery interval in minutes** to 3 minutes.

4. **sp_configure "number of device", 0, "default"**

   Resets the value for **number of devices** to the Adaptive Server default.

Comments

- Any user can execute **sp_configure** to display information about parameters and their current values, but not to modify parameters. System Administrators can execute **sp_configure** to change the values of most configuration parameters. Only System Security Officers can execute certain parameters. These are listed under "Permissions" in this section.
- When you execute **sp_configure** to modify a dynamic parameter:
  - The configuration and run values are updated.
  - The configuration file is updated.
  - The change takes effect immediately.
- When you execute **sp_configure** to modify a static parameter:
  - The configuration value is updated.
  - The configuration file is updated.
  - The change takes effect only when you restart Adaptive Server.
- When issued with no parameters, **sp_configure** displays a report of all configuration parameters by group, their current values, their default values, the value (if applicable) to which they have most recently been set, and the amount of memory used by this setting:
  - The *default* column in the report displays the value Adaptive Server is shipped with. If you do not explicitly reconfigure a parameter, it retains its default value.
  - The *memory used* column displays the amount of memory used by the parameter at its current value. Some related parameters draw from the same memory pool. For instance, the memory used for **stack size** and **stack guard size** is already accounted for in the memory used for **number of user connections**. If you added the memory used by each of these parameters separately, it would total more than the amount actually used. In the *memory used* column, parameters that "share" memory with other parameters are marked with a hash mark (#).
  - The *config_value* column displays the most recent value to which the configuration parameter has been set with **sp_configure**.
  - The *run_value* column displays the value being used by Adaptive Server. It changes after you modify a parameter's value with **sp_configure** and, for static parameters, after you

restart Adaptive Server. This is the value stored in *syscurconfigs.value*.

➤ ***Note***

If the server uses a case-insensitive sort order, **sp_configure** with no parameters returns a list of all configuration parameters and groups in alphabetical order with no grouping displayed.

- Each configuration parameter has an associated display level. There are three display levels:
  - The "basic" level displays only the most basic parameters. It is appropriate for very general server tuning.
  - The "intermediate" level displays parameters that are somewhat more complex, as well as showing you all the "basic" parameters. This level is appropriate for a moderately complex level of server tuning.
  - The "comprehensive" level displays all parameters, including the most complex ones. This level is appropriate for users who do highly detailed server tuning.

  The default display level is "comprehensive". Setting one of the other display levels lets you work with a subset of the configuration parameter, shortening the amount of information displayed by **sp_configure**.

  The syntax for showing your current display level is:

  ```
  sp_displaylevel
  ```

- For information on the individual configuration parameters, see the *System Administration Guide*.

Permissions

Any user can execute **sp_configure** to display information about parameters and their current values.

Only System Administrators and System Security Officers can execute **sp_configure** to modify configuration parameters.

Only System Security Officers can execute **sp_configure** to modify values for:

allow procedure grouping
allow select on syscomments.text
allow updates
audit queue size

auditing
current audit table
remote access
suspend auditing when full
systemwide password expiration

System Administrators can modify all other parameters.

Tables Used

master..sysdevices, *master..sysconfigures*, *master..syscurconfigs*, *master..sysdatabases*, *master..sysdevices*, *master..sysindexes*, *master..syslanguages*, *master..sysmessages*, *master..sysobjects*, *master..sysservers*

See Also

Commands	set
System procedures	**sp_addlanguage**,**sp_audit**, **sp_dboption**, **sp_displaylevel**, **sp_droplanguage**, **sp_helpconfig**, **sp_modifylogin**, **sp_monitorconfig**

sp_copy_all_qplans

Function

Copies all plans for one abstract plan group to another group.

Syntax

sp_copy_all_qplans ***src_group, dest_group***

Parameters

src_group – is the name of the source abstract plan group.

dest_group – is the name of the abstract plan group to which the plans are to be copied.

Examples

1. sp_copy_all_qplans dev_plans, ap_stdin

Copies all of the abstract plans in the *dev_plans* group to the *ap_stdin* group.

Comments

- The destination group must exist before you can copy plans into it. It may contain plans.
- **sp_copy_all_qplans** calls **sp_copy_qplan** for each plan in the source group. Each plan is copied as a separate transaction, so any problem that keeps **sp_copy_all_qplans** from completing does not affect the plans that have already been copied.
- **sp_copy_qplan** prints messages when it cannot copy a particular abstract plan. You also see these messages when running **sp_copy_all_qplans**.
- If the query text for a plan in the destination group exactly matches the query text in the source group and the user ID is the same, the plan is not copied, and a message giving the plan ID is sent to the user, but the copying process continues with the next plan in the source group.
- Copying a very large number of abstract plans can take considerable time, and also requires space on the *system* segment in the database and space to log the changes to the database. Use **sp_spaceused** to check the size of *sysqueryplans*, and **sp_helpsegment** for the *system* and *logsegment* to check the space available.

Permissions

Any user can execute **sp_copy_all_qplans** to copy an abstract plan that he or she owns. Only the System Administrator or Database Owner can copy plans that are owned by other users.

Tables Used

sysattributes, sysqueryplans

See Also

System procedures	**sp_copy_qplan**, **sp_help_qpgroup**

sp_copy_qplan

Function

Copies one abstract plan to an abstract plan group.

Syntax

sp_copy_qplan *src_id, dest_group*

Parameters

src_id – is the ID of the abstract plan to copy.

dest_group – is the name of the destination abstract plan group.

Examples

```
1. sp_copy_qplan 2140534659, ap_stdin
```

Comments

- The destination group must exist before you can copy an abstract plan into it. You do not need to specify a source group, since plans are uniquely identified by the plan ID.
- A new plan ID is generated when the plan is copied. The plan retains the ID of the user who created it, even if the System Administrator or Database Owner copies the plan. To assign a different user ID, a System Administrator or Database Owner can use **sp_export_qpgroup** and **sp_import_qpgroup**.
- If the query text for a plan in the destination group exactly matches the query text in the source group and the user ID, the plan is not copied, and a message giving the plan IDs is sent to the user.
- To copy all of the plans in an abstract plan group, use **sp_copy_all_qplans**.

Permissions

Any user can execute **sp_copy_qplan** to copy a plan that he or she owns. Only the System Administrator or Database Owner can copy plans that are owned by other users.

Tables Used

sysattributes, *sysqueryplans*

See Also

System procedures	**sp_copy_all_qplans**, **sp_help_qpgroup**, **sp_help_qplan**, **sp_import_qpgroup**

sp_countmetadata

Function

Displays the number of indexes, objects, or databases in Adaptive Server.

Syntax

```
sp_countmetadata "configname" [, dbname]
```

Parameters

configname – is either **"open indexes"**, **"open objects"**, or **"open databases"**.

dbname – is the name of the database on which to run **sp_countmetadata**. If no database name is given, **sp_countmetadata** provides a total count for all databases.

Examples

1. sp_countmetadata "open objects"

```
There are 283 user objects in all database(s),
requiring 117.180 Kbytes of memory. The 'open
objects' configuration parameter is currently set
to a run value of 500.
```

Reports on the number of user objects in Adaptive Server. Use this value to set the number of objects allowed in the database, plus space for additional objects and temporary tables. For example:

```
sp_configure "number of open objects", 310
```

2. sp_countmetadata "open indexes", pubs2

```
There are 21 user indexes in pubs2 database(s),
requiring 8.613 kbytes of memory. The 'open
indexes' configuration parameter is currently set
to 600.
```

Reports on the number of indexes in Adaptive Server.

Comments

- **sp_countmetadata** displays the number of indexes, objects, or databases in Adaptive Server, including the number of system databases such as *model* and *tempdb*.

- Avoid running **sp_countmetadata** during Adaptive Server peak times. It can cause contention on the *sysindexes*, *sysobjects*, and *sysdatabases* system tables.
- You can run **sp_countmetadata** on a specified database if you want information on a particular database. However, when configuring caches for indexes, objects, or databases, run **sp_countmetadata** without the *database_name* option.
- The information on memory returned by **sp_countmetadata** can vary by platform. For example, a database on Adaptive Server for Windows NT could have a different **sp_countmetadata** result than the same database on Sun Solaris. Information on the number of user indexes, objects, or databases should be consistent, however.
- **sp_countmetadata** does not include temporary tables in its calculation. Add 5 percent to the **open objects** value and 10 percent to the **open indexes** value to accommodate temporary tables.
- If you specify a nonunique fragment of **"open indexes"**, **"open objects"**, or **"open databases"** for *configname*, **sp_countmetadata** returns a list of matching configuration parameter names with their configured values and current values. For example:

  ```
  sp_countmetadata "open"
  ```

```
Configuration option is not unique.
 option_name                    config_value run_value
 ------------------------------ ------------ -----------
 curread change w/ open cursors            1           1
 number of open databases                 12          12
 number of open indexes                  500         500
 number of open objects                  500         500
 open index hash spinlock ratio          100         100
 open index spinlock ratio               100         100
 open object spinlock ratio              100         100
```

Permissions

Only a System Administrator or the Database Owner can execute **sp_countmetadata.**

Tables Used

master..sysdatabases, *sysindexes*, *sysobjects*

See Also

System procedures	**sp_configure**, **sp_helpconfig, sp_monitorconfig**

sp_cursorinfo

Function

Reports information about a specific cursor or all cursors that are active for your session.

Syntax

```
sp_cursorinfo [{cursor_level | null}] [, cursor_name]
```

Parameters

cursor_level | **null** – is the level at which Adaptive Server returns information for the cursors. You can specify the following for *cursor_level*:

Level	Types of Cursors
N	Any cursors declared inside stored procedures at a specific procedure nesting level. You can specify any positive number for its level.
0	Any cursors declared outside stored procedures.
-1	Any cursors from either of the above. You can substitute any negative number for this level.

If you want information about cursors with a specific *cursor_name*, regardless of cursor level, specify **null** for this parameter.

cursor_name – is the specific name for the cursor. Adaptive Server reports information about all active cursors that use this name at the *cursor_level* you specify. If you omit this parameter, Adaptive Server reports information about all the cursors at that level.

Examples

1. sp_cursorinfo 0, authors_crsr

```
Cursor name 'authors_crsr' is declared at nesting level '0'.
The cursor id is 327681
The cursor has been successfully opened 1 times.
The cursor was compiled at isolation level 0.
The cursor is not open.
The cursor will remain open when a transaction is commited or
rolled back.
The number of rows returned for each FETCH is 1.
The cursor is read only.
There are 3 columns returned by this cursor.
The result columns are:
Name = 'au_id', Table = 'authors', Type = ID,
     Length = 11 (read only)
Name = 'au_lname', Table = 'authors', Type = VARCHAR,
Length = 40 (read only)
Name = 'au_fname', Table = 'authors', Type = VARCHAR,
Length = 20 (read only)
```

Displays the information about the cursor named *authors_crsr* at level 0.

2. sp_cursorinfo null, author_sales

```
Cursor name 'author_sales' is declared on procedure 'au_sales'.
Cursor name 'author_sales' is declared at nesting level '1'.
The cursor id is 327682
The cursor has been successfully opened 1 times.
The cursor was compiled at isolation level 1.
The cursor is currently scanning at a nonzero isolation level.
The cursor is positioned after the last row.
The cursor will be closed when a transaction is commited or
rolled back.
The number of rows returned for each FETCH is 1.
The cursor is updatable.
There are 3 columns returned by this cursor.
The result columns are:
Name = 'title_id', Table = 'titleauthor', Type = ID,
      Length = 11 (updatable)
Name = 'title', Table = 'titles', Type = VARCHAR,
      Length = 80 (updatable)
Name = 'total_sales', Table = 'titles', Type = INT (updatable)
```

Displays the information about any cursors named *author_sales* declared by a user across all levels.

Comments

- If you do not specify either *cursor_level* or *cursor_name*, Adaptive Server displays information about all active cursors. Active cursors are those declared by you and allocated by Adaptive Server.
- Adaptive Server reports the following information about each cursor:
  - The cursor name, its nesting level, its cursor ID, and the procedure name (if it is declared in a stored procedure).
  - The number of times the cursor has been opened.
  - The isolation level (0, 1, or 3) in which it was compiled and in which it is currently scanning (if open).
  - Whether the cursor is open or closed. If the cursor is open, it indicates the current cursor position and the number of rows fetched.
  - Whether the open cursor will be closed if the cursor's current position is deleted.
  - Whether the cursor will remain open or be closed if the cursor's current transaction is committed or rolled back.
  - The number of rows returned for each fetch of that cursor.
  - Whether the cursor is updatable or read-only.
  - The number of columns returned by the cursor. For each column, it displays the column name, the table name or expression result, and whether it is updatable.

  The output from **sp_cursorinfo** varies, depending on the status of the cursor. In addition to the information listed, **sp_cursorinfo** displays the **showplan** output for the cursor. For more information about **showplan**, see the *Performance and Tuning Guide*.

Permissions

Any user can execute **sp_cursorinfo**.

Tables Used

sysobjects

See Also

Commands	**declare cursor**, **set**

sp_dboption

Function

Displays or changes database options.

Syntax

sp_dboption [*dbname*, *optname*, {true | false}]

Parameters

dbname – is the name of the database in which the option is to be set. You must be using *master* to execute **sp_dboption** with parameters (that is, to change a database option). You cannot, however, change option settings in the *master* database.

optname – is the name of the option to be set. Adaptive Server understands any unique string that is part of the option name. Use quotes around the option name if it is a keyword or includes embedded blanks or punctuation.

{**true** | **false**} – **true** to turn the option on, **false** to turn it off.

Examples

1. sp_dboption

```
Settable database options

 database_options
 ------------------------
 abort tran on log full
 allow nulls by default
 auto identity
 dbo use only
 ddl in tran
 identity in nonunique index
 no chkpt on recovery
 no free space acctg
 read only
 select into/bulkcopy/pllsort
 single user
 trunc log on chkpt
 trunc. log on chkpt.
 unique auto_identity index
```

Displays a list of the database options.

2.
```
use pubs2
go
master..sp_dboption pubs2, "read", true
go
checkpoint
go
```

   Makes the database *pubs2* read only. The **read** string uniquely identifies the **read only** option from among all available database options. Note the use of quotes around the keyword **read**.

3.
```
pubs2..sp_dboption pubs2, "read", false
go
checkpoint
go
```

   Makes the database *pubs2* writable again.

4.
```
use pubs2
go
master..sp_dboption pubs2, "select into", true
go
checkpoint
go
```

   Allows **select into**, **bcp** and parallel sort operations on tables in the *pubs2* database. The **select into** string uniquely identifies the **select into/ bulkcopy** option from among all available database options. Note that quotes are required around the option because of the embedded space.

5.
```
use mydb
go
master..sp_dboption mydb, "auto identity", true
go
checkpoint
go
```

   Automatically defines 10-digit IDENTITY columns in new tables created in *mydb*. The IDENTITY column, *SYB_IDENTITY_COL*, is defined in each new table that is created without specifying either a primary key, a **unique** constraint, or an IDENTITY column.

6.
```
use master
go
sp_dboption mydb, "identity in nonunique index",
true
go
use mydb
go
checkpoint
go
```

   Automatically includes an IDENTITY column in the *mydb* tables' index keys, provided these tables already have an IDENTITY column. All indexes created on the tables will be internally unique.

7.
```
use master
go
sp_dboption pubs2, "unique auto_identity index",
true
go
use pubs2
go
checkpoint
go
```

   Automatically includes an IDENTITY column with a unique, nonclustered index for new tables in the *pubs2* database.

Comments

- The *master* database option settings cannot be changed.
- To display a list of database options, execute **sp_dboption** with no parameters from inside the *master* database.
- For a report on which database options are set in a particular database, execute **sp_helpdb**.
- The Database Owner or System Administrator can set or unset particular database options for all new databases by executing **sp_dboption** on *model*.
- After **sp_dboption** has been executed, the change does not take effect until the **checkpoint** command is issued in the database for which the option was changed.

Database Options

- The **abort tran on log full** option determines the fate of a transaction that is running when the last-chance threshold is crossed in the log segment of the specified database. The default value is **false**,

meaning that the transaction is suspended and is awakened only when space has been freed. If you change the setting to **true**, all user queries that need to write to the transaction log are killed until space in the log has been freed.

- Setting the **allow nulls by default** option to **true** changes the default value of a column from **not null** to **null**, in compliance with the SQL standards. The Transact-SQL default value for a column is **not null**, meaning that null values are not allowed in a column unless **null** is specified in the **create table** or **alter table** column definition. **allow nulls by default true** reverses this.
- While the **auto identity** option is set to **true** (on), a 10-digit IDENTITY column is defined in each new table that is created without specifying either a **primary** key, a **unique** constraint, or an IDENTITY column. The column is not visible when you select all columns with the **select *** statement. To retrieve it, you must explicitly mention the column name, *SYB_IDENTITY_COL*, in the select list.

  To set the precision of the automatic IDENTITY column, use the **size of auto identity column** configuration parameter.

  Though you can set **auto identity** to **true** in *tempdb*, it is not recognized or used, and temporary tables created there do not automatically include an IDENTITY column.

  For a report on indexes in a particular table that includes the IDENTITY column, execute **sp_helpindex**.
- While the **dbo use only** option is set to **true** (on), only the database's owner can use the database.
- When the **ddl in tran option** is set to **true** (on), you can use certain data definition language commands in transactions. If **ddl in tran** is **true** in a particular database, commands such as **create table**, **grant**, and **alter table** are allowed inside transactions in that database. If **ddl in tran** is **true** in the *model* database, the commands are allowed inside transactions in all databases created after **ddl in tran** was set in *model*.

◆ ***WARNING!***

Data definition language commands hold locks on system tables such as *sysobjects*. Avoid using them inside transactions; if you must use them, keep the transactions short.

Using any data definition language commands on *tempdb* within transactions may cause your system to grind to a halt. Always leave ddl in tran set to false in *tempdb*.

- Table 7-16 lists the commands that can be used inside a user-defined transaction when the **ddl in tran** option is set to **true**:

Table 7-16: DDL commands allowed in transactions

alter table (clauses other than **partition** and **unpartition** are allowed)	**create default**	**drop default**	**grant**
	create index	**drop index**	**revoke**
	create procedure	**drop procedure**	
	create rule	**drop rule**	
	create schema	**drop table**	
	create table	**drop trigger**	
	create trigger	**drop view**	
	create view		

- Table 7-17 lists the commands that cannot be used inside a user-defined transaction under any circumstances:

Table 7-17: DDL commands not allowed in transactions

alter database	**dump database**	**select into**
alter table...partition	**dump transaction**	**truncate table**
alter table...unpartition	**drop database**	**update statistics**
create database	**load transaction**	
disk init	**load database**	

  In addition, system procedures that create temporary tables or change the *master* database cannot be used inside user-defined transactions.

- The **identity in nonunique index** option automatically includes an IDENTITY column in a table's index keys, so that all indexes created on the table are unique. This database option makes logically nonunique indexes internally unique, and allows these indexes to be used to process updatable cursors and isolation level 0 reads.

The table must already have an IDENTITY column for the **identity in nonunique index** option to work, either from a **create table** statement or by setting the **auto identity** database option to **true** before creating the table.

Use **identity in nonunique index** if you plan to use cursors and isolation level 0 reads on tables with nonunique indexes. A unique index ensures that the cursor will be positioned at the correct row the next time a **fetch** is performed on that cursor. If you plan to use cursors on tables with unique indexes and any isolation level, you may want to use the **unique auto_identity index** option.

For a report on indexes in a particular table that includes the IDENTITY column, execute **sp_helpindex**.

- The **no free space acctg** option suppresses free-space accounting and execution of threshold actions for the non-log segments. This speeds recovery time because the free-space counts are not recomputed for those segments.
- The **no chkpt on recovery** option is set to **true** (on) when an up-to-date copy of a database is kept. In these situations, there is a "primary" and a "secondary" database. Initially, the primary database is dumped and loaded into the secondary database. Then, at intervals, the transaction log of the primary database is dumped and loaded into the secondary database.

  If this option is set to **false** (off), the default condition, a checkpoint record is added to a database after it is recovered when you restart Adaptive Server. This checkpoint, which ensures that the recovery mechanism will not be rerun unnecessarily, changes the sequence number and causes a subsequent load of the transaction log from the primary database to fail.

  Setting this option to **true** (on) for the secondary database causes it not to get a checkpoint from the recovery process so that subsequent transaction log dumps from the primary database can be loaded into it.
- The **read only** option means that users can retrieve data from the database, but cannot modify any data.
- Setting the **select into/bulkcopy/pllsort** option to **true** (on) enables the use of **writetext**, **select into** a permanent table, "fast" bulk copy into a table that has no indexes or triggers, using **bcp** or the bulk copy library routines, and parallel sort. A transaction log dump cannot recover these minimally logged operations, so **dump transaction** to a

dump device is prohibited. After non-logged operations are completed, set **select into/bulk copy/pllsort** to **false** (off) and issue **dump database**.

Issuing the **dump transaction** statement after unlogged changes have been made to the database with **select into**, bulk copy, or parallel sort produces an error message instructing you to use **dump database** instead. (The **writetext** command does not have this protection.)

You do not have to set the **select into/bulkcopy/pllsort** option to **true** in order to **select into** a temporary table, since *tempdb* is never recovered. The option need not be set to **true** in order to run **bcp** on a table that has indexes, because tables with indexes are always copied with the slower version of bulk copy and are logged.

- When **single user** is set to **true**, only one user at a time can access the database (single-user mode).

  You cannot set **single user** to **true** in a user database from within a stored procedure or while users have the database open. You cannot set **single user** to **true** for *tempdb*.

- The **trunc log on chkpt** option means that if the transaction log has more than 50 rows of committed transactions, the transaction log is truncated (the committed transactions are removed) every time the **checkpoint** checking process occurs (usually more than once per minute). When the Database Owner runs **checkpoint** manually, however, the log is **not** truncated. It may be useful to turn this option on while doing development work, to prevent the log from growing.

  While the **trunc log on chkpt** option is on, **dump transaction** to a dump device is prohibited, since dumps from the truncated transaction log cannot be used to recover from a media failure. Issuing the **dump transaction** statement produces an error message instructing you to use **dump database** instead.

- When the **unique auto_identity index** option is set to **true**, it adds an IDENTITY column with a unique, nonclustered index to new tables. By default, the IDENTITY column is a 10-digit numeric datatype, but you can change this default with the **size of auto identity column** configuration parameter. As with **auto identity**, the IDENTITY column is not visible when you select all columns with the **select *** statement. To retrieve it, you must explicitly mention the column name, *SYB_IDENTITY_COL*, in the select list.

If you need to use cursors or isolation level 0 reads with nonunique indexes, use the **identity in nonunique index** option.

Though you can set **unique auto_identity index** to **true** in *tempdb*, it is not recognized or used, and temporary tables created there do not automatically include an IDENTITY column with a unique index.

- For more information on database options, see the *System Administration Guide.*

Permissions

Only a System Administrator or the Database Owner can execute **sp_dboption** with parameters to change database options. A user aliased to the Database Owner cannot execute **sp_dboption** to change database options. Any user can execute **sp_dboption** with no parameters to view database options.

Tables Used

master.dbo.sysdatabases, master.dbo.sysmessages, master.dbo.sysprocesses, sysobjects

See Also

Commands	**checkpoint**, **select**
System procedures	**sp_configure**, **sp_helpdb**, **sp_helpindex**, **sp_helpjoins**
Utility commands	**bcp**

sp_dbrecovery_order

Function

Specifies the order in which user databases are recovered and lists the user-defined recovery order of a database or all databases.

Syntax

```
sp_dbrecovery_order
   [database_name [, rec_order [, force]]]
```

Parameters

database_name – The name of the database being assigned a recovery order or the database whose user-defined recovery order is to be listed.

rec_order – The order in which the database is to be recovered. A *rec_order* of **-1** deletes a specified database from the user-defined recovery sequence.

force – allows the user to insert a database into an existing recovery sequence without putting it at the end.

Examples

1. sp_dbrecovery_order pubs2, 1

Makes the *pubs2* database the first user database to be recovered following a system failure.

2. sp_dbrecovery_order pubs3, 3, force

Inserts the *pubs3* database into third position in a user-defined recovery sequence. If another database was initially in third position, it is moved to fourth position, and all databases following it are moved accordingly.

3. sp_dbrecovery_order pubs2, -1

Removes the *pubs2* database from the user-defined recovery sequence. Subsequently, *pubs2* will be recovered after all databases with a user-specified recovery order have recovered.

4. sp_dbrecovery_order

Lists the current recovery order of all databases with a recovery order assigned through **sp_dbrecovery_order**.

Comments

- You must be in the *master* database to use **sp_dbrecovery_order** to enter or modify a user-specified recovery order. You can list the user-defined recovery order of databases from any database.
- To change the user-defined recovery position of a database, use **sp_dbrecovery_order** to delete the database from the recovery sequence, then use **sp_dbrecovery_order** to insert it into a new position.
- System databases are always recovered before user databases. The system databases and their recovery order are:
  - *master*
  - *model*
  - *tempdb*
  - *sybsystemdb*
  - *sybsecurity*
  - *sybsystemprocs*
- If no database is assigned a recovery order through **sp_dbrecovery_order**, all user databases are recovered in order, by database ID, after system databases.
- If *database_name* is specified, but no *rec_order* is given, **sp_dbrecovery_order** shows the user-defined recovery position of the specified database.
- If *database_name* is not specified, **sp_dbrecovery_order** lists the recovery order of all databases with a user-assigned recovery order.
- The order of recovery assigned through **sp_dbrecovery_order** must be consecutive, starting with **1** and containing no gaps between values. The first database assigned a recovery order must be assigned a *rec_order* of **1**. If three databases have been assigned a recovery order of **1**, **2**, and **3**, you cannot assign the next database a recovery order of **5**.

Permissions

Only a System Administrator can execute **sp_dbrecovery_order**.

Tables Used

master..sysattributes

sp_dbremap

Function

Forces Adaptive Server to recognize changes made by **alter database**. Run this procedure only when instructed to do so by an Adaptive Server message.

Syntax

sp_dbremap ***dbname***

Parameters

dbname – is the name of the database in which the **alter database** command was interrupted.

Examples

1. sp_dbremap sample_db

An **alter database** command changed the database *sample_db*. This command makes the changes visible to Adaptive Server.

Comments

- If an **alter database** statement issued on a database that is in the process of being dumped is interrupted, Adaptive Server prints a message instructing the user to execute **sp_dbremap**.
- Any changes to *sysusages* during a database or transaction dump are not copied into active memory until the dump completes, to ensure that database mapping does not change during the dump. Running **alter database** makes changes to system tables on the disk immediately. In-memory allocations cannot be changed until a dump completes. This is why **alter database** pauses.

  When you execute **sp_dbremap**, it must wait until the dump process completes.
- If you are instructed to run **sp_dbremap**, but do not do it, the space you have allocated with **alter database** does not become available to Adaptive Server until the next restart.

Permissions

Only a System Administrator or Database Owner can execute **sp_dbremap**.

Tables Used

master.dbo.sysdatabases, sysobjects

See Also

Commands	**alter database**, **dump database**, **dump transaction**

sp_defaultloc

(Component Integration Services only)

Function

Defines a default storage location for objects in a local database.

Syntax

```
sp_defaultloc dbname, {"defaultloc"| NULL}
   [, "defaulttype"]
```

Parameters

dbname – is the name of a database being mapped to a remote storage location. The database must already have been defined by a **create database** statement. You cannot map system databases to a remote location.

defaultloc – is the remote storage location to which the database is being mapped. To direct the server to delete an existing default mapping for a database, supply NULL for this parameter. The value of *defaultloc* must end in a period (.), as follows:

```
server.dbname.owner.
```

defaulttype – is one of the values that specify the format of the object named by *object_loc*. Table 7-18 describes the valid values. Enclose the *defaulttype* value in quotes.

Table 7-18: Allowable values for defaulttype

Value	Description
table	Indicates that the object named by *object_loc* is a table accessible to a remote server. This value is the default for *defaulttype*.
view	Indicates that the object named by *object_loc* is a view managed by a remote server, processed as a table.
rpc	Indicates that the object named by *object_loc* is an RPC managed by a remote server; processes the result set from the RPC as a read-only table.

Examples

1. **sp_defaultloc pubs, "SYBASE.pubs.dbo.", "table"**
   create table pubs.dbo.book1 (bridges char(15))

   sp_defaultloc defines the remote storage location *pubs.dbo.* in the remote server named SYBASE. It maps the database *pubs* to the remote location. A "**create table** book1" statement would create a table named *book1* at the remote location. A **create existing table** statement for *bookN* would require that *pubs.dbo.bookN* already exist at the remote location, and information about table *bookN* would be stored in the local table *bookN*.

2. **sp_defaultloc pubs, NULL**

   Removes the mapping of the database *pubs* to a remote location.

3. **sp_defaultloc ticktape,"wallst.nasdaq.dbo.", "rpc"**
   create existing table sybase (bestbuy integer)

   Identifies the remote storage location *wallst.nasdaq.dbo* where "wallst" is the value provided for *server_name*, "nasdaq" is provided for *database*, and "dbo" is provided for *owner*. The RPC *sybase* must already exist at the remote location. A "**create existing table** sybase" statement would store information about the result set from RPC *sybase* in local table *ticktape*. The result set from RPC *sybase* is regarded as a read-only table. Inserts, updates and deletes are not supported for RPCs.

Comments

- **sp_defaultloc** defines a default storage location for tables in a local database. It maps table names in a database to a remote location. It permits the user to establish a default for an entire database, rather than issue an **sp_addobjectdef** command before every **create table** and **create existing table** command.
- When *defaulttype* is **table**, **view**, or **rpc**, the *defaultloc* parameter takes the form:

  server_name.dbname.owner.

  - Note that the *defaultloc* specification ends in a period (.).
  - *server_name* represents a server already added to *sysservers* by **sp_addserver**. The *server_name* parameter is required.
  - *dbname* might not be required. Some server classes do not support it.
  - *owner* should always be provided to avoid ambiguity. If it is not provided, the remote object actually referenced could vary,

depending on whether the external login corresponds to the remote object owner.

- Issue **sp_defaultloc** before any **create table** or **create existing table** statement. When either statement is used, the server uses the *sysattributes* table to determine whether any table mapping has been specified for the object about to be created or defined. If the mapping has been specified, a **create table** statement directs the table to be created at the location specified by *object_loc*. A **create existing table** statement stores information about the existing remote object in the local table.
- If you issue **sp_defaultloc** on *defaulttype* **view** and then issue **create table**, Component Integration Services creates a new table, not a view, on the remote server.
- Changing the default location for a database does not affect tables that have previously been mapped to a different default location.
- After tables in the database have been created, all future references to tables in *dbname* (by **select**, **insert**, **delete,** and **update**) are mapped to the correct location.

Permissions

Any user can execute **sp_defaultloc**.

Tables Used

master.dbo.sysservers, master.dbo.spt_values, master.dbo.sysattributes

See Also

Commands	**create existing table**, **create table**
System procedures	**sp_addobjectdef**,**sp_addserver**, **sp_help**, **sp_helpserver**

sp_depends

Function

Displays information about database object dependencies—the view(s), trigger(s), and procedure(s) in the database that depend on a specified table or view, and the table(s) and view(s) in the database on which the specified view, trigger, or procedure depends.

Syntax

sp_depends *objname*

Parameters

objname – is the name of the table, view, stored procedure, or trigger to be examined for dependencies. You cannot specify a database name. Use owner names if the object owner is not the user running the command and is not the Database Owner.

Examples

1. sp_depends sysobjects

Lists the database objects that depend on the table *sysobjects*.

2. sp_depends titleview

```
Things that the object references in the current
database.

object          type        updated selected
--------------  ----------- ------- -----
dbo.authors     user table  no      no
dbo.titleauthor user table  no      no
dbo.titles      user table  no      no

Things inside the current database that reference
the object.

object          type
------------    ---------------
dbo.tview2      view
```

Lists the database objects that depend on the *titleview* view, and the database objects on which the *titleview* view depends.

3. sp_depends "mary.titles"

Lists the database objects that depend on the *titles* table owned by the user “mary”. The quotes are needed, since the period is a special character.

Comments

- Executing **sp_depends** lists all objects in the current database that depend on *objname*, and on which *objname* depends. For example, views depend on one or more tables and can have procedures or other views that depend on them. An object that references another object is dependent on that object. References to objects outside the current database are not reported.
- The **sp_depends** procedure determines the dependencies by looking at the *sysdepends* table.

  If the objects were created out of order (for example, if a procedure that uses a view was created before the view was created), no rows exist in *sysdepends* for the dependencies, and **sp_depends** does not report the dependencies.
- The *updated* and *selected* columns in the report from **sp_depends** are meaningful if the object being reported on is a stored procedure or trigger. The values for the *updated* column indicate whether the stored procedure updates the object. The *selected* column indicates whether the object is being used for a read cursor or a data modification statement.
- **sp_depends** follows these Adaptive Server rules for finding objects:
  - If the user does not specify an owner name, and the user executing the command owns an object with the specified name, that object is used.
  - If the user does not specify an owner name, and the user does not own an object of that name, but the Database Owner does, the Database Owner's object is used.
  - If neither the user nor the Database Owner owns an object of that name, the command reports an error condition, even if an object exists in the database with that object name, but with a different owner.
  - If both the user and the Database Owner own objects with the specified name, and the user wants to access the Database Owner's object, the name must be specified, as in *dbo.objectname*.
- Objects owned by database users other than the user executing a command and the Database Owner must always be qualified with the owner's name, as in example 3.

Permissions

Any user can execute **sp_depends**.

Tables Used

master.dbo.spt_values, *master.dbo.sysmessages*, *sysdepends*, *sysobjects*, *sysusers*

See Also

Commands	**create procedure**, **create table**, **create view**, **execute**
System procedures	**sp_help**

sp_deviceattr

Function

Changes the **dsync** setting of an existing database device file.

Syntax

sp_deviceattr *logicalname*, *optname*, *optvalue*

Parameters

logicalname – is the name of an existing database device. The device can be stored on either an operating system file or a raw partition, but the **dsync** setting is ignored for raw partitions.

optname – is the name of the setting to change. Currently, the only acceptable value for *optname* is **dsync**.

optvalue – can be either "true" or "false."

Examples

1. sp_deviceattr file_device1, dsync, true

Sets **dsync** on for the device named "file_device1."

Comments

- For database devices stored on UNIX files, **dsync** determines whether updates to the device take place directly on the storage media, or are buffered by the UNIX file system.

  When **dsync** is on, writes to the database device occur directly to the physical storage media, and Adaptive Server can recover data on the device in the event of a system failure.

  When **dsync** is off, writes to the database device may be buffered by the UNIX file system. The UNIX file system may mark an update as being completed, even though the physical media has not yet been modified. In the event of a system failure, there is no guarantee that requests to update data have ever taken place on the physical media, and Adaptive Server may be unable to recover the database.
- After using **sp_deviceattr** to change the **dsync** setting, you must reboot Adaptive Server before the change takes affect.
- **dsync** is always on for the master device file. You cannot change the **dsync** setting for a master device file with **sp_deviceattr**.

- The **dsync** value should be turned off only when the databases on the device need not be recovered after a system failure. For example, you may consider turning **dsync** off for a device that stores only the *tempdb* database.
- Adaptive Server ignores the **dsync** setting for devices stored on raw partitions—updates to those devices are never buffered, regardless of the **dsync** setting.
- **dsync** is not used on the Windows NT platform.

Permissions

The user executing **sp_deviceattr** must have permission to update the *sysdevices* table.

Tables Used

sysdevices

See Also

System procedures	**sp_helpdevice**

sp_diskdefault

Function

Specifies whether or not a database device can be used for database storage if the user does not specify a database device or specifies **default** with the **create database** or **alter database** commands.

Syntax

```
sp_diskdefault logicalname, {defaulton | defaultoff}
```

Parameters

logicalname – is the logical name of the device as given in *master.dbo.sysdevices.name*. The device must be a database device rather than a dump device.

defaulton | **defaultoff** – **defaulton** designates the database device as a default database device; **defaultoff** designates that the specified database device is not a default database device.

Use **defaulton** after adding a database device to the system with **disk init**. Use **defaultoff** to change the default status of the master device (which is designated as a default device when Adaptive Server is first installed).

Examples

```
1. sp_diskdefault master, defaultoff
```

The master device is no longer used by **create database** or **alter database** for default storage of a database.

Comments

- A default database device is one that is used for database storage by **create database** or **alter database** if the user does not specify a database device name or specifies the keyword **default**.
- You can have multiple default devices. They are used in the order they appear in the *master.dbo.sysdevices* table (that is, alphabetical order). When the first default device is filled, the second default device is used, and so on.
- When you first install Adaptive Server, the master device is the only default database device.

➤ ***Note***

Once you initialize devices to store user databases, use **sp_diskdefault** to turn off the master device's default status. This prevents users from accidentally creating databases on the master device and simplifies recovery of the *master* database.

- To find out which database devices are default database devices, execute **sp_helpdevice**.

Permissions

Only a System Administrator can execute **sp_diskdefault**.

Tables Used

master.dbo.sysdevices, *sysobjects*

See Also

Commands	**alter database**, **create database**, **disk init**
System procedures	**sp_helpdevice**

sp_displayaudit

Function

Displays the status of audit options.

Syntax

```
sp_displayaudit ["procedure" | "object" | "login" |
   "database" | "global" | "default_object" |
   "default_procedure" [, "name"]]
```

Parameters

procedure – displays the status of audit options for the specified stored procedure or trigger. If you do not specify a value for *name*, **sp_displayaudit** displays the active audit options for all procedures and triggers in the current database.

object – displays the status of audit options for the specified table or view. If you do not specify a value for *name*, **sp_displayaudit** displays the active audit options for all tables and views in the current database.

login – displays the status of audit options for the specified user login. If you do not specify a value for *name*, **sp_displayaudit** displays the active audit options for all logins in the *master* database.

database – displays the status of audit options for the specified database. If you do not specify a value for *name*, **sp_displayaudit** displays the active audit options for all databases on the server.

global – displays the status of the specified global audit option. If you do not specify a value for *name*, **sp_displayaudit** displays the active audit options for all procedures and triggers in the current database.

default_object – displays the default audit options that will be used for any new table or view created on the specified database. If you do not specify a value for *name*, **sp_displayaudit** displays the default audit options for all databases with active default audit settings.

default_procedure – displays the default audit options that will be used for any new procedure or trigger created on the specified database. If you do not specify a value for *name*, **sp_displayaudit** displays the default audit options for all databases with active default audit settings.

name – is the information for the specified parameter, as described in the following table:

Parameter	Value for *name*
procedure	Procedure or trigger name
object	Table or view name
login	User login
database	Database name
global	Global audit option
default_object	Database name
default_procedure	Database name

Examples

1. sp_displayaudit

```
Procedure/Trigger    Audit Option   Value Database
 -----------------   -------------- ----- --------------------
 dbo.sp_altermessage exec_procedure on    sybsystemprocs
 dbo.sp_help         exec_procedure on    sybsystemprocs
 dbo.sp_who          exec_procedure on    sybsystemprocs
No databases currently have default sproc/trigger auditing
enabled.
No objects currently have auditing enabled.
No databases currently have default table/view auditing enabled.
No logins currently have auditing enabled.
No databases currently have auditing enabled.

Option Name                    Value
------------------------------ ------------------------------
adhoc                          off
dbcc                           off
disk                           off
errors                         off
login                          off
logout                         off
navigator_role                 off
oper_role                      off
replication_role               off
rpc                            off
sa_role                        off
security                       off
sso_role                       off
```

When no parameter is specified, the status of each category and all auditing options is displayed.

2. sp_displayaudit "procedure"

```
Procedure/Trigger    Audit Option   Value Database
 -----------------   -------------- ----- ---------------------
 dbo.sp_altermessage exec_procedure on    sybsystemprocs
 dbo.sp_help         exec_procedure on    sybsystemprocs
 dbo.sp_who          exec_procedure on    sybsystemprocs
```

When no procedure name is specified, the status of all procedure audit options is displayed.

3. sp_displayaudit "procedure", "sp_who"

```
Procedure/Trigger Audit Option    Value Database
----------------- -------------- ----- ---------------------
dbo.sp_who        exec_procedure  on    sybsystemprocs
```

When you specify a name for the procedure, only the status of that procedure is displayed.

4. sp_displayaudit "global"

```
Option Name                    Value
------------------------------ ------------------------------
adhoc                          off
dbcc                           off
disk                           off
errors                         off
login                          off
logout                         off
navigator_role                 off
oper_role                      off
replication_role               off
rpc                            off
sa_role                        off
security                       off
sso_role                       off
```

When no global audit option is specified, the status of all global audit options is displayed.

Comments

- **sp_displayaudit** displays the status of audit options.

- The following table shows the valid auditing options for each parameter:

Object Type Parameter	Valid Auditing Options
procedure	exec_procedure, exec_trigger
object	delete, func_obj_access, insert, reference, select, update
login	all, cmdtext, table_access, view_access
database	alter, bcp, bind, create, dbaccess, drop, dump, func_dbaccess, grant, load, revoke, setuser, truncate, unbind
global	adhoc, dbcc, disk, errors, login, logout, navigator_role, oper_role, replication_role, rpc, sa_role, security, sso_role
default_object	delete, func_obj_access, insert, reference, select, update
default_procedure	exec_procedure, exec_trigger

- You cannot specify a value for name unless you first specify an object type parameter.
- For information on setting up auditing, see the *System Administration Guide.*

Permissions

Only a System Security Officer can execute **sp_displayaudit.**

Tables Used

sysauditoptions, sysdatabases, syslogins, sysobjects

See Also

System procedures	sp_audit
Utility commands	bcp

sp_displaylevel

Function

Sets or shows which Adaptive Server configuration parameters appear in **sp_configure** output.

Syntax

```
sp_displaylevel [loginame [, level]]
```

Parameters

loginame – is the Adaptive Server login of the user for whom you want to set or show the display level.

level – sets the display level to one of the following:

"basic" display level shows just the most basic configuration parameters. This level is appropriate for very general server tuning.

"intermediate" display level shows configuration parameters that are somewhat more complex, as well as all the "basic" level parameters. This level is appropriate for moderately complex server tuning.

"comprehensive" display level shows all configuration parameters, including the most complex ones. This level is appropriate for highly detailed server tuning.

Examples

1. sp_displaylevel

```
The current display level for login 'sa' is
'comprehensive'.
```

Shows the current display level for the user who invoked **sp_displaylevel**.

2. sp_displaylevel jerry

```
The current display level for login 'jerry' is
'intermediate'.
```

Shows the current display level for the user "jerry".

3. sp_displaylevel jerry, comprehensive

```
The display level for login 'jerry' has been
changed to 'comprehensive'.
```

Sets the display level to “comprehensive” for the user “jerry”.

Permissions

Only a System Administrator can execute **sp_displaylevel** to set the display level for another user. Any user can execute **sp_displaylevel** to set and show his or her own display level.

Tables Used

master..sysattributes

See Also

System procedures	**sp_configure**

sp_displaylogin

Function

Displays information about a login account. Also displays information about the hierarchy tree above or below the login account when you so specify.

Syntax

```
sp_displaylogin [loginame [, expand_up | expand_down]]
```

Parameters

loginame – is the user login account about which you want information if it is other than your own. You must be a System Security Officer or System Administrator to get information about someone else's login account.

expand_up – specifies that Adaptive Server display all roles in the role hierarchy that contain the *loginame* role.

expand_down – specifies that Adaptive Server display all roles in the role hierarchy that are contained by the *loginame* role.

Examples

1. sp_displaylogin

```
Suid: 1
Loginame: sa
Fullname:
Default Database: master
Default Language:
Configured Authorization:
    sa_role (default ON)
    sso_role (default ON)
    oper_role (default ON)
Locked: NO
Date of Last Password Change: Nov 16 1994 10:08AM
```

Displays information about your server login account.

2. sp_displaylogin susanne

```
Suid: 12
Loginame: susanne
Fullname:
Default Database: pubs2
Default Language:
Configured Authorization:
    supervisor (default OFF)
Locked: NO
Date of Last Password Change: May 12 1997 11:09AM
```

Displays information about the login account "susanne". The information displayed varies, depending on the role of the user executing **sp_displaylogin**.

3. sp_displaylogin pillai, expand_up

Displays information about all roles containing the role of the login account "pillai". The information displayed varies, depending on the role of the user executing **sp_displaylogin**.

Comments

- **sp_displaylogin** displays configured roles, so even if you have made a role inactive with the **set** command, it is displayed.
- When you use **sp_displaylogin** to get information about your own account, you do not need to use the *loginame* parameter. **sp_displaylogin** displays your server user ID, login name, full name, any roles that have been granted to you, date of last password change, default database, default language, and whether your account is locked.
- If you are a System Security Officer or System Administrator, you can use the *loginame* parameter to access information about any account.

Permissions

Only a System Administrator or a System Security Officer can execute **sp_displaylogin** with the *loginame* and **expand** parameters to get information about other users' login accounts. Any user can execute **sp_displaylogin** to get information about his or her own login account.

Tables Used

master.dbo.sysloginroles, *master.dbo.syslogins*, *master.dbo.syssrvroles*, *sysobjects*

See Also

Stored procedures	**sp_activeroles**, **sp_displayroles**, **sp_helprotect**, **sp_modifylogin**

sp_displayroles

Function

Displays all roles granted to another role, or displays the entire hierarchy tree of roles in table format.

Syntax

sp_displayroles [*grantee_name* [, *mode*]]

Parameters

grantee_name – is the login name of a user whose roles you want information about, or the name of a role you want information about.

mode – is **expand_up** or **expand_down**. **expand_up** shows the role hierarchy tree for the parent levels and **expand_down** shows the role hierarchy tree for the child levels.

Examples

1. sp_displayroles

```
Role Name
------------------------------
supervisor_role
```

Displays all roles granted to the user issuing the command.

2. sp_displayroles "supervisor_role"

```
Role Name
------------------------------
clerk
```

Displays all roles granted to *supervisor_role*.

3. sp_displayroles susanne, expand_down

```
Role Name         Parent Role Name       Level
----------------- ---------------------- ------
supervisor_role   NULL                        1
clerk_role       supervisor_role             2
```

Displays the active roles granted to login “susanne” and the roles below it in the hierarchy.

4. sp_displayroles "intern_role", expand_up

Displays the active roles granted to *intern_role* and the roles above it in the hierarchy.

Comments

- When you specify the optional parameter **expand_up** or **expand_down** all directly granted roles contained by or containing the specified role name are displayed.

Permissions

Only a System Administrator or a System Security Officer can execute **sp_displayroles** to display information on roles activated by any other user. Any user can execute **sp_displayroles** to see his or her own active roles.

Tables Used

master.dbo.sysattributes, master.dbo.syssrvroles, master.dbo.syslogins, master.dbo.sysloginroles

See Also

Commands	**alter role, create role, drop role, grant, revoke, set**
System procedures	**sp_activeroles**, **sp_displaylogin**, **sp_helprotect**, **sp_modifylogin**

sp_dropalias

Function

Removes the alias user name identity established with **sp_addalias**.

Syntax

```
sp_dropalias loginame
```

Parameters

loginame – is the name (in *master.dbo.syslogins*) of the user who was aliased to another user.

Examples

```
1. sp_dropalias victoria
```

Assuming that "victoria" was aliased (for example, to the Database Owner) in the current database, this statement drops "victoria" as an aliased user from the database.

Comments

- Executing the **sp_dropalias** procedure deletes an alternate *suid* mapping for a user from the *sysalternates* table.
- When a user's alias is dropped, he or she no longer has access to the database for which the alias was created.

Permissions

Only the Database Owner or a System Administrator can execute **sp_dropalias**.

Tables Used

sysalternates, *sysobjects*

See Also

Commands	**use**
System procedures	**sp_addalias**, **sp_adduser**, **sp_changedbowner**, **sp_droplogin**, **sp_dropuser**, **sp_helpuser**

sp_drop_all_qplans

Function

Deletes all abstract plans in an abstract plan group.

Syntax

```
sp_drop_all_qplans name
```

Parameters

name – is the name of the abstract plan group from which to drop all plans.

Examples

```
1. sp_drop_all_qplans dev_test
```

Comments

- To drop individual plans, use **sp_drp_qplan**.
- To see the names of abstract plan groups in the current database, use **sp_help_qpgroup**.
- **sp_drop_all_qplans** silently drops all plans in the group that belong to the specified user, or all plans in the group, if it is executed by a System Administrator or Database Owner.

Permissions

Any user can execute **sp_drop_all_qplans** to drop plans that he or she owns. Only a System Administrator or Database Owner can drop plans owned by other users.

Tables Used

sysattributes, *sysqueryplans*

See Also

System procedures	**sp_drop_qplan**, **sp_help_qpgroup**

sp_dropdevice

Function

Drops an Adaptive Server database device or dump device.

Syntax

```
sp_dropdevice logicalname
```

Parameters

logicalname – is the name of the device as listed in *master.dbo.sysdevices.name*.

Examples

1. `sp_dropdevice tape5`

   Drops the device named *tape5* from Adaptive Server.

2. `sp_dropdevice fredsdata`

   Drops the database device named *fredsdata* from Adaptive Server. The device must not be in use by any database.

Comments

- The **sp_dropdevice** procedure drops a device from Adaptive Server, deleting the device entry from *master.dbo.sysdevices*.
- **sp_dropdevice** does not remove a file that is being dropped as a database device; it makes the file inaccessible to Adaptive Server. Use operating system commands to delete a file after using **sp_dropdevice**.

Permissions

Only a System Administrator can execute **sp_dropdevice**.

Tables Used

master.dbo.sysdatabases, *master.dbo.sysdevices*, *master.dbo.sysusages*, *sysobjects*

See Also

Commands	**drop database**
System procedures	**sp_addumpdevice**, **sp_helpdb**, **sp_helpdevice**

sp_dropengine

Function

Drops an engine from a specified engine group or, if the engine is the last one in the group, drops the engine group.

Syntax

sp_dropengine *engine_number, engine_group*

Parameters

engine_number – is the number of the engine you are dropping from the group. Values are between 0 and a maximum equal to the number of configured online engines, minus one.

engine_group – is the name of the engine group from which to drop the engine.

Examples

1. sp_dropengine 2, DS_GROUP

This statement drops engine number 2 from the group called *DS_GROUP*. If it is the last engine in the group, the group is also dropped.

Comments

- **sp_dropengine** can be invoked only from the *master* database.
- If *engine_number* is the last engine in *engine_group*, Adaptive Server also drops *engine_group*.
- The *engine_number* you specify must exist in *engine_group*.

Permissions

Only a System Administrator can execute **sp_dropengine**.

Tables Used

sysattributes

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropexeclass, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe**

sp_dropexeclass

Function

Drops a user-defined execution class.

Syntax

```
sp_dropexeclass classname
```

Parameters

classname – is the name of the user-defined execution class to be dropped.

Examples

```
1. sp_dropexeclass 'DECISION'
```

This statement drops the user-defined execution class *DECISION*.

Comments

- An execution class helps define the execution precedence used by Adaptive Server to process tasks. For more information on execution classes and execution attributes, see the *Performance and Tuning Guide*.
- *classname* must not be bound to any client application, login, or stored procedure. Unbind the execution class first, using **sp_unbindexeclass**, then drop the execution class, using **sp_dropexeclass**.
- You cannot drop system-defined execution classes.

Permissions

Only a System Administrator can execute **sp_dropexeclass**.

Tables Used

sysattributes

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_setpsexe, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**

sp_dropextendedproc

Function

Removes an extended stored procedure (ESP).

Syntax

```
sp_dropextendedproc esp_name
```

Parameters

esp_name – is the name of the extended stored procedure to be dropped.

Examples

```
1. sp_dropextendedproc xp_echo
```

Removes *xp_echo*.

Comments

- **sp_dropextendedproc** must be executed from the *master* database.
- The *esp_name* is case sensitive. It must precisely match the name with which the ESP was created.

Permissions

Only a System Administrator can execute **sp_dropextendedproc.**

Tables Used

master.dbo.syscomments, sysobjects

See Also

Commands	**drop procedure**
System procedures	**sp_addextendedproc, sp_freedll, sp_helpextendedproc**

sp_dropexternlogin

(Component Integration Services only)

Function

Drops the definition of a remote login previously defined by **sp_addexternlogin.**

Syntax

```
sp_dropexternlogin remote_server [, login_name]
```

Parameters

remote_server – is the name of the remote server from which the local server is dropping account access. The *remote_server* is known to the local server by an entry in the *master.dbo.sysservers* table.

login_name – is a login account known to the local server. If *login_name* is not specified, the current account is used. *login_name* must exist in the *master.dbo.syslogins* table.

Examples

1. `sp_dropexternlogin JOBSERV, sa`

   Drops the definition of an external login to the remote server JOBSERV from *login_name* "sa".

2. `sp_dropexternlogin CIS1012, bobj`

   Drops the definition of an external login to the remote server CIS1012 from "bobj". Only the "bobj" account and the "sa" account can add or modify a remote login for "bobj".

Comments

- **sp_dropexternlogin** drops the definition of a remote login previously defined to the local server by **sp_addexternlogin.**
- You cannot execute **sp_dropexternlogin** from within a transaction.
- The *remote_server* must be defined to the local server by **sp_addserver.**
- To add and drop local server users, use the system procedures **sp_addlogin** and **sp_droplogin.**

Permissions

Only *login_name* or a System Administrator can execute **sp_dropexternlogin**.

Tables Used

master.dbo.sysattributes, master.dbo.sysservers

See Also

System procedures	**sp_addlogin**, **sp_addexternlogin**, **sp_addobjectdef**, **sp_addserver**,**sp_droplogin**, **sp_dropobjectdef**, **sp_helpdb**, **sp_helpserver**

sp_dropglockpromote

Function

Removes lock promotion values from a table or database.

Syntax

```
sp_dropglockpromote {"database" | "table"}, objname
```

Parameters

database | **table** – specifies whether to remove the lock promotion thresholds from a database or table. The quotes are required because these are Transact-SQL keywords.

objname – is the name of the table or database from which to remove the lock promotion thresholds.

Examples

```
1. sp_dropglockpromote "table", titles
```

Removes the lock promotion values from *titles*. Lock promotion for *titles* now uses the database or server-wide values.

Comments

- Use **sp_dropglockpromote** to drop lock promotion values set with **sp_setpglockpromote**.
- When you drop a database's lock promotion thresholds, tables that do not have lock promotion thresholds configured will use the server-wide values.
- When a table's values are dropped, Adaptive Server uses the database's lock promotion thresholds if they are configured or the server-wide values if they are not.
- Server-wide values can be changed with **sp_setpglockpromote**, but cannot be dropped.

Permissions

Only a System Administrator can execute **sp_dropglockpromote**.

Tables Used

master.dbo.sysattributes, *sysobjects*

See Also

System procedures	**sp_configure**, **sp_setpglockpromote**

sp_dropgroup

Function

Drops a group from a database.

Syntax

```
sp_dropgroup grpname
```

Parameters

grpname – is the name of a group in the current database.

Examples

1.
```
sp_changegroup accounting, martha
sp_changegroup "public", george
sp_dropgroup purchasing
```

   The "purchasing" group has merged with the "accounting" group. These commands move "martha" and "george", members of the "purchasing" group, to other groups before dropping the group. The group name "public" is quoted because "public" is a reserved word.

Comments

- Executing **sp_dropgroup** drops a group name from a database's *sysusers* table.
- You cannot drop a group if it has members. You must execute **sp_changegroup** for each member before you can drop the group.

Permissions

Only the Database Owner, a System Administrator, or a System Security Officer can execute **sp_dropgroup**.

Tables Used

master.dbo.syssrvroles, *sysobjects*, *sysprotects*, *sysusers*

See Also

Commands	**grant**, **revoke**, **use**
System procedures	**sp_addgroup**, **sp_adduser**, **sp_changegroup**, **sp_dropuser**, **sp_helpgroup**

sp_dropkey

Function

Removes from the *syskeys* table a key that had been defined using **sp_primarykey**, **sp_foreignkey**, or **sp_commonkey**.

Syntax

```
sp_dropkey keytype, tabname [, deptabname]
```

Parameters

keytype – is the type of key to be dropped. The *keytype* must be **primary**, **foreign**, or **common**.

tabname – is the name of the key table or view that contains the key to be dropped.

deptabname – specifies the name of the second table in the relationship, if the *keytype* is **foreign** or **common**. If the *keytype* is **primary**, this parameter is not needed, since **primary** keys have no dependent tables. If the *keytype* is **foreign**, this is the name of the primary key table. If the *keytype* is **common**, give the two table names in the order in which they appear with **sp_helpkey**.

Examples

1. `sp_dropkey primary, employees`

   Drops the primary key for the *employees* table. Any foreign keys that were dependent on the primary key for *employees* are also dropped.

2. `sp_dropkey common, employees, projects`

   Drops the common keys between the *employees* and *projects* tables.

3. `sp_dropkey foreign, titleauthor, titles`

   Drops the foreign key between the *titleauthor* and *titles* tables.

Comments

- Executing **sp_dropkey** deletes the specified key from *syskeys*. Only the owner of a table can drop a key from that table.
- Keys are created to make explicit a logical relationship that is implicit in your database design. This information can be used by an application.

- Dropping a primary key automatically drops any foreign keys associated with it. Dropping a foreign key has no effect on a primary key specified on that table.
- Executing **sp_commonkey, sp_primarykey,** or **sp_foreignkey** adds the key to the *syskeys* system table. To display a report on the keys that have been defined, execute **sp_helpkey.**

Permissions

Only the owner of *tabname* can execute **sp_dropkey.**

Tables Used

syskeys, *sysobjects*

See Also

System procedures	**sp_commonkey**, **sp_foreignkey**, **sp_helpkey**, **sp_primarykey**

sp_droplanguage

Function

Drops an alternate language from the server and removes its row from *master.dbo.syslanguages*.

Syntax

```
sp_droplanguage language [, dropmessages]
```

Parameters

language – is the official name of the language to be dropped.

dropmessages – drops all Adaptive Server system messages in *language*. You cannot drop a language with associated system messages without also dropping its messages.

Examples

1. `sp_droplanguage french`

   This command drops French from the available alternate languages, if there are no associated messages.

2. `sp_droplanguage french, dropmessages`

   This command drops French from the available alternate languages, if there are associated messages.

Comments

- Executing **sp_droplanguage** drops a language from a list of alternate languages by deleting its entry from the *master.dbo.syslanguages* table.
- If you try to drop a language that has system messages, the request fails unless you supply the **dropmessages** parameter.

Permissions

Only a System Administrator can execute **sp_droplanguage**.

Tables Used

master.dbo.syslanguages, *master.dbo.sysmessages*, *sysobjects*

See Also

System procedures	**sp_addlanguage**, **sp_helplanguage**

sp_droplogin

Function

Drops an Adaptive Server user login by deleting the user's entry from *master.dbo.syslogins*.

Syntax

sp_droplogin *loginame*

Parameters

loginame – is the name of the user, as listed in *master.dbo.syslogins*.

Examples

1. sp_droplogin victoria

Drops the "victoria" login from Adaptive Server.

Comments

- Executing **sp_droplogin** drops a user login from Adaptive Server, deleting the user's entry from *master.dbo.syslogins*.
- Adaptive Server reuses a dropped login's server user ID, which compromises accountability. You can avoid dropping accounts entirely and, instead, use **sp_locklogin** to lock any accounts that will no longer be used.

  If you need to drop logins, be sure to audit these events (using **sp_audit**) so that you have a record of them.
- **sp_droplogin** deletes all resource limits associated with the dropped login.
- **sp_droplogin** fails if the login to be dropped is a user in any database on the server. Use **sp_dropuser** to drop the user from a database. You cannot drop a user from a database if that user owns any objects in the database.
- If the login to be dropped is a System Security Officer, **sp_droplogin** verifies that at least one other unlocked System Security Officer's account exists. If not, **sp_droplogin** fails. Similarly, **sp_droplogin** ensures that there is always at least one unlocked System Administrator account.

Permissions

Only a System Administrator or a System Security Officer can execute **sp_droplogin**.

Tables Used

master.dbo.sysloginroles, master.dbo.syslogins, master.dbo.sysprocesses, sysobjects

See Also

System procedures	**sp_addlogin**, **sp_audit**, **sp_changedbowner**, **sp_dropalias**, **sp_dropuser**, **sp_helpuser**, **sp_locklogin**

sp_dropmessage

Function

Drops user-defined messages from *sysusermessages*.

Syntax

```
sp_dropmessage message_num [, language]
```

Parameters

message_num – is the message number of the message to be dropped. Message numbers must have a value of 20000 or higher.

language – is the language of the message to be dropped.

Examples

1. `sp_dropmessage 20002, french`

   Removes the French version of the message with the number 20002 from *sysusermessages*.

Comments

- The *language* parameter is optional. If included, only the message with the indicated *message_num* in the indicated language is dropped. If you do not specify a *language*, all messages with the indicated *message_num* are dropped.

Permissions

Only the Database Owner, a System Administrator, or the user who created the message being dropped can execute **sp_dropmessage**.

Tables Used

master.dbo.syslanguages, *sysobjects*, *sysusermessages*

See Also

System procedures	**sp_addmessage**, **sp_getmessage**

sp_drop_qpgroup

Function

Drops an abstract plan group.

Syntax

```
sp_drop_qpgroup group
```

Parameters

group – is the name of the abstract plan group to drop.

Examples

```
1. sp_drop_qpgroup dev_test
```

Comments

- You cannot drop the default groups, *ap_stdin* and *ap_stdout*.
- You cannot drop a group that contains plans. To drop all of the plans in a a group, use **sp_drop_all_qplans.** To see a list of groups and the number of plans they contain, use **sp_help_qpgroup**.
- **sp_drop_qpgroup** cannot be run in a transaction.

Permissions

Only a System Administrator or Database Owner can execute **sp_drop_qpgroup**.

Tables Used

sysattributes, sysqueryplans

See Also

System procedures	**sp_drop_all_qplans**, **sp_help_qpgroup**

sp_drop_qplan

Function

Drops an abstract plan.

Syntax

```
sp_drop_qplan id
```

Parameters

id – is the ID of the abstract plan to drop.

Examples

```
1. sp_drop_qplan 1760009301
```

The abstract plan with the specified ID is dropped.

Comments

- To find the ID of a plan, use **sp_help_qpgroup**, **sp_help_qplan**, or **sp_find_qplan**. Plan IDs are also returned by **create plan** and are included in **showplan** output.
- To drop all abstract plans in a group, use **sp_drop_all_qplans**.

Permissions

Any user can execute **sp_drop_qplan** to drop a plan he or she owns. Only the System Administrator or the Database Owner can drop plans owned by other others.

Tables Used

sysqueryplans

See Also

Commands	**create plan**
System procedures	**sp_drop_all_qplans**, **sp_find_qplan**, **sp_help_qpgroup**, **sp_help_qplan**

sp_dropobjectdef

(Component Integration Services only)

Function

Deletes the external storage mapping provided for a local object.

Syntax

```
sp_dropobjectdef "object_name"
```

Parameters

object_name has the form *dbname.owner.object*, where:

dbname is the name of the database containing the object whose storage location you are dropping. *dbname* is optional; if present, it must be the current database, and the *owner* or a placeholder is required.

owner is the name of the owner of the object whose storage location you are dropping. *owner* is optional; it is required if *dbname* is specified.

object is the name of the local table for which external storage mapping is to be dropped.

Examples

1. `sp_dropobjectdef "personnel.dbo.colleges"`

   Deletes the entry from *sysattributes* that provided the external storage mapping for a table known to the server as the *colleges* table in database *personnel*.

2. `sp_dropobjectdef "andrea.fishbone"`

   Deletes the entry from *sysattributes* that provided the external storage mapping for the *andrea.fishbone* object, where *andrea* is the *owner* and the local table name is *fishbone*.

Comments

- **sp_dropobjectdef** deletes the external storage mapping provided for a local object. It replaces **sp_droptabledef**.
- Use **sp_dropobjectdef** after dropping a remote table with **drop table**.
- Dropping a table does not remove the mapping information from the *sysattributes* table if it was added using **sp_addobjectdef**. It must be explicitly removed using **sp_dropobjectdef**.

- The *object_name* can be in any of these forms:
  - *object*
  - *owner.object*
  - *dbname..object*
  - *dbname.owner.object*

Permissions

Only the Database Owner or a System Administrator can execute **sp_dropobjectdef**. Only a System Administrator can execute **sp_dropobjectdef** to remove mapping information for another user's object.

Tables Used

sysobjects, sysattributes, sysusers

See Also

Commands	**create existing table**, **create table, drop table**
System procedures	**sp_addobjectdef**

sp_dropremotelogin

Function

Drops a remote user login.

Syntax

```
sp_dropremotelogin remoteserver [, loginame
   [, remotename] ]
```

Parameters

remoteserver – is the name of the server that has the remote login to be dropped.

loginame – is the local server's user name that is associated with the remote server in the *sysremotelogins* table.

remotename – is the remote user name that gets mapped to *loginame* when logging in from the remote server.

Examples

1. `sp_dropremotelogin GATEWAY`

   Drops the entry for the remote server named GATEWAY.

2. `sp_dropremotelogin GATEWAY, churchy`

   Drops the entry for mapping remote logins from the remote server GATEWAY to the local user named "churchy".

3. `sp_dropremotelogin GATEWAY, churchy, pogo`

   Drops the login for the remote user "pogo" on the remote server GATEWAY that was mapped to the local user named "churchy".

Comments

- Executing **sp_dropremotelogin** drops a user login from a remote server, deleting the user's entry from *master.dbo.sysremotelogins*.
- For a more complete discussion on remote logins, see **sp_addremotelogin**.
- To add and drop local server users, use the system procedures **sp_addlogin** and **sp_droplogin**.

Permissions

Only a System Administrator can execute **sp_dropremotelogin**.

Tables Used

master.dbo.sysremotelogins, master.dbo.sysservers, sysobjects

See Also

System procedures	**sp_addlogin**, **sp_addremotelogin**, **sp_addserver**, **sp_droplogin**, **sp_helpremotelogin**, **sp_helpserver**

sp_drop_resource_limit

Function

Removes one or more resource limits from Adaptive Server.

Syntax

```
sp_drop_resource_limit {name, appname }
   [, rangename, limittype, enforced, action, scope]
```

Parameters

name – is the Adaptive Server login to which the limit applies. To drop resource limits that apply to all users of a particular application, specify the *appname* and a *name* of NULL.

appname – is the application to which the limit applies. To drop resource limits that apply to all applications used by the specified login, specify the login name and an *appname* of NULL. To drop a limit that applies to a particular application, specify the application name that the client program passes to the Adaptive Server in the login packet.

rangename – is the time range during which the limit is enforced. This must be an existing time range stored in the *systimeranges* system table or NULL to delete all resource limits for the specified *name*, *appname*, *limittype*, *action*, and *scop*e, without regard to *rangename*.

limittype – is the type of resource being limited. This must be one of the following:

Limit Type	Description
row_count	Drops only limits that restrict the number of rows a query can return.
elapsed_time	Drops only limits that restrict the number of seconds that a query batch or transaction can run.
io_cost	Drops only limits that restrict actual or estimated query processing cost.
NULL	Drops all resource limits with the specified *name, appname, rangename,* enforcement time, *action,* and *scope,* without regard to *limittyp*e.

enforced – determines whether the limit is enforced prior to or during query execution. The following table lists the valid values for each limit type:

Enforced Code	Description	Limit Type
1	Drops only limits for which action is taken when the estimated cost of execution exceeds the specified limit.	**io_cost**
2	Drops only limits for which action is taken when the actual row count, elapsed time, or cost of execution exceeds the specified limit.	**row_count** **elapsed_time** **io_cost**
3	Drops only limits for which action is taken when either the estimated cost (1) or the actual cost (2) exceeds the specified limit.	**io_cost**
NULL	Drops all resource limits with the specified *name*, *appname*, *rangename*, *limittype*, and *scope*, without regard to when the *action* is enforced.	

action – is the action taken when the limit is exceeded. This must be one of the following:

Action Code	Description
1	Drops only limits that issue a warning.
2	Drops only limits that abort the query batch.
3	Drops only limits that abort the transaction.
4	Drops only limits that kill the session.
NULL	Drops all resource limits with the specified *name*, *appname*, *rangename*, *limittype*, enforcement time, and *scope*, without regard to the *action* they take.

scope – is the scope of the limit. This must be one of the following:

Scope Code	Description
1	Drops only limits that apply to queries.
2	Drops only limits that apply to query batches.

Scope Code	Description
4	Drops only limits that apply to transactions.
6	Drops only limits that apply to both query batches and transactions.
NULL	Drops all resource limits with the specified *name*, *appname*, *rangename*, *limittype*, enforcement time, and *action*, without regard to their *scope*.

Examples

1. **`sp_drop_resource_limit joe, payroll, friday_afternoon, io_cost, 2, 4, 1`**

   Drops the single resource limit that kills the session whenever joe's use of the *payroll* application runs a query during the *friday_afternoon* time range that results in excessive execution-time I/O cost.

➤ ***Note***

If no resource limit matches these selection criteria, **sp_drop_resource_limit** returns without error.

2. **`sp_drop_resource_limit joe, payroll`**

   Drops all limits that apply to joe's use of the *payroll* application.

3. **`sp_drop_resource_limit joe`**

   Drops all limits that apply to the user "joe".

4. **`sp_drop_resource_limit NULL, payroll`**

   Drops all resource limits that apply to the *payroll* application.

5. **`sp_drop_resource_limit NULL, payroll, NULL, NULL, NULL, 4, NULL`**

   Drops all resource limits on the *payroll* application whose action is to kill the session.

Comments

- Use the **sp_help_resource_limit** system procedure to determine which resource limits apply to a given user, application, or time of day.
- When you use **sp_droplogin** to drop an Adaptive Server login, all resource limits associated with that login are also dropped.

- The deletion of a resource limit causes the limits for each session for that login and / or application to be rebound at the beginning of the next query batch for that session.
- For more information on resource limits, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_drop_resource_limit.**

Tables Used

master..sysresourcelimits, master..systimeranges, master..spt_limit_types

See Also

System procedures	**sp_add_resource_limit**, **sp_help_resource_limit**, **sp_modify_resource_limit**

sp_droprowlockpromote

Function

Removes row lock promotion threshold values from a database or table.

Syntax

```
sp_droprowlockpromote {"database" | "table"}, objname
```

Parameters

database | **table** – specifies whether to remove the row lock promotion thresholds from a database or table.

objname – is the name of the database or table from which to remove the row lock promotion thresholds.

Examples

```
1. sp_droprowlockpromote "table", "sales"
```

Removes the row lock promotion values from the *sales* table. Lock promotion for *sales* now uses the database or server-wide values.

Comments

- Use **sp_droprowlockpromote** to drop row lock promotion values set with **sp_setrowlockpromote.**
- When you drop a database's row lock promotion thresholds, datarows-locked tables that do not have row lock promotion thresholds configured use the server-wide values. Use **sp_configure** to check the value of the row lock promotion configuration parameters.
- When a table's row lock promotion values are dropped, Adaptive Server uses the database's row lock promotion thresholds, if they are configured, or the server-wide values, if no thresholds are set for the database.
- To change the lock promotion thresholds for a database, you must be using the *master* database. To change the lock promotion thresholds for a table in a database, you must be using the database where the table resides.
- Server-wide values can be changed with **sp_setrowlockpromote**. This changes the values in the row lock promotion configuration

parameters, so there is no corresponding server option for **sp_droprowlockpromote**.

Permissions

Only a System Administrator can execute **sp_droprowlockpromote**.

Tables Used

master.dbo.sysattributes, sysobjects

See Also

System procedures	**sp_setrowlockpromote**

sp_dropsegment

Function

Drops a segment from a database or unmaps a segment from a particular database device.

Syntax

```
sp_dropsegment segname, dbname [, device]
```

Parameters

segname – is the name of the segment to be dropped.

dbname – is the name of the database from which the segment is to be dropped.

device – is the name of the database device from which the segment *segname* is to be dropped. This parameter is optional, except when the system segment *system*, *default*, or *logsegment* is being dropped from a database device.

Examples

1. `sp_dropsegment indexes, pubs2`

   This command drops the segment *indexes* from the *pubs2* database.

2. `sp_dropsegment indexes, pubs2, dev1`

   This command unmaps the segment *indexes* from the database device *dev1*.

Comments

- You can drop a segment if it is not referenced by any table or index in the specified database.
- If you do not supply the optional argument *device*, the segment is dropped from the specified database. If you do supply a *device* name, the segment is no longer mapped to the named database device, but the segment is not dropped.
- Dropping a segment drops all thresholds associated with that segment.
- When you unmap a segment from one or more devices, Adaptive Server drops any thresholds that exceed the total space on the

segment. When you unmap the *logsegment* from one or more devices, Adaptive Server recalculates the last-chance threshold.

- **sp_placeobject** changes future space allocations for a table or index from one segment to another, and removes the references from the original segment. After using **sp_placeobject**, you can drop the original segment name with **sp_dropsegment**.
- For the system segments *system*, *default*, and *logsegment*, you must specify the device name from which you want the segments dropped.

Permissions

Only the Database Owner or a System Administrator can execute **sp_dropsegment**.

Tables Used

master.dbo.spt_values, *sysdatabases*, *sysdevices*, *sysindexes*, *sysobjects*, *syssegments*, *systhresholds*, *sysusages*

See Also

System procedures	**sp_addsegment**, **sp_addthreshold**, **sp_helpsegment**, **sp_helpthreshold**, **sp_placeobject**

sp_dropserver

Function

Drops a server from the list of known servers or drops remote logins and external logins in the same operation.

Syntax

```
sp_dropserver server [, droplogins]
```

Parameters

server – is the name of the server to be dropped.

droplogins – indicates that any remote logins for *server* should also be dropped.

Examples

1. `sp_dropserver GATEWAY`

   This command drops the remote server GATEWAY.

2. `sp_dropserver RDBAM_ALPHA, droplogins`

   Drops the entry for the remote server RDBAM_ALPHA and drops all remote logins and external logins for that server.

Comments

- Executing **sp_dropserver** drops a server from the list of known servers by deleting the entry from the *master.dbo.sysservers* table.
- Running **sp_dropserver** on a server that has associated entries in the *master.dbo.sysremotelogins* table results in an error message stating that you must drop the remote users before you can drop the server. To drop all remote logins for a server when dropping the server, use **droplogins**.
- Running **sp_dropserver** without **droplogins** against a server that has associated entries in the *sysattributes* table results in an error. You must drop the remote logins and external logins before you can drop the server.
- The checks against *sysattributes* for external logins and for default mapping to a server apply when Component Integration Services is configured.

Permissions

Only a System Security Officer can execute **sp_dropserver.**

Tables Used

master.dbo.sysremotelogins, master.dbo.sysservers, sysobjects

See Also

System procedures	**sp_addserver**, **sp_dropremotelogin**, **sp_helpremotelogin**, **sp_helpserver**

sp_dropthreshold

Function

Removes a free-space threshold from a segment.

Syntax

```
sp_dropthreshold dbname, segname, free_space
```

Parameters

dbname – is the database from which you are dropping the threshold. This must be the name of the current database.

segname – is the segment whose free space is monitored by the threshold. Use quotes when specifying the "default" segment.

free_space – is the number of free pages at which the threshold is crossed.

Examples

```
1. sp_dropthreshold mydb, segment1, 200
```

Removes a threshold from *segment1* of *mydb*. You must specify the database, segment, and amount of free space to identify the threshold.

Comments

- You cannot drop the last-chance threshold from the log segment.
- You can use the **no free space acctg** option of **sp_dboption** as an alternative to **sp_dropthreshold**. This option disables free-space accounting on non-log segments. You cannot disable free-space accounting on log segments.

Permissions

Only the Database Owner or a System Administrator can execute **sp_dropthreshold**.

Tables Used

sysobjects, *syssegments*, *systhresholds*

See Also

System procedures	**sp_addthreshold**, **sp_dboption**, **sp_helpthreshold**, **sp_thresholdaction**

sp_drop_time_range

Function

Removes a user-defined time range from Adaptive Server.

Syntax

```
sp_drop_time_range name
```

Parameters

name – is the name of the time range to be dropped.

Examples

1. `sp_drop_time_range evenings`

Removes the "evenings" time range.

Comments

- You cannot remove the "at all times" time range.
- You cannot drop a time range if a resource limit exists for that time range.
- Dropping a time range does not affect the active time ranges for sessions currently in progress.
- For more information on time ranges, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_drop_time_range**.

Tables Used

master..systimeranges

See Also

System procedures	**sp_add_resource_limit**, **sp_add_time_range**, **sp_modify_time_range**

sp_droptype

Function

Drops a user-defined datatype.

Syntax

```
sp_droptype typename
```

Parameters

typename – is the name of a user-defined datatype that you own.

Examples

1. `sp_droptype birthday`

Drops the user-defined datatype named *birthday*.

Comments

- Executing **sp_droptype** deletes a user-defined datatype from *systypes*.
- A user-defined datatype cannot be dropped if it is referenced by tables or another database object.

Permissions

Only the Database Owner or datatype owner can execute **sp_droptype**.

Tables Used

syscolumns, *sysobjects*, *systypes*, *sysusers*

See Also

Datatypes	"System and User-Defined Datatypes"
System procedures	**sp_addtype**, **sp_rename**

sp_dropuser

Function

Drops a user from the current database.

Syntax

sp_dropuser ***name_in_db***

Parameters

name_in_db – is the user's name in the current database's *sysusers* table.

Examples

1. sp_dropuser albert

Drops the user "albert" from the current database. The user "albert" can no longer use the database.

Comments

- **sp_dropuser** drops a user from the current database by deleting the user's row from *sysusers*.
- You cannot drop a user who owns objects in the database.
- You cannot drop a user who has granted permissions to other users.
- You cannot drop the Database Owner from a database.
- If other users are aliased to the user being dropped, their aliases are also dropped. They no longer have access to the database.
- You cannot drop a user from a database if the user owns a stored procedure that is bound to an execution class in that database. See **sp_bindexeclass**.

Permissions

Only the Database Owner, a System Administrator, or a System Security Officer can execute **sp_dropuser.**

Tables Used

master.dbo.spt_values, *sysalternates*, *syscolumns*, *sysobjects*, *sysprotects*, *syssegments*, *systhresholds*, *systypes*, *sysusers*

See Also

Commands	**grant**, **revoke**, **use**
System procedures	**sp_addalias**, **sp_adduser**, **sp_bindexeclass**, **sp_droplogin**

sp_dumpoptimize

Function

Specifies the amount of data dumped by Backup Server during the dump database operation.

Syntax

```
sp_dumpoptimize [ 'archive_space =
   {maximum | minimum | default }' ]
sp_dumpoptimize [ 'reserved_threshold =
   {nnn | default }' ]
sp_dumpoptimize [ 'allocation_threshold =
   {nnn | default }' ]
```

Parameters

archive_space – specifies the amount of the database you want dumped.

maximum – dumps the whole database without determining which pages are allocated or not. The total space used by the archive image or images is equal to the size of the database. Using this option has the same effect as using the options **reserved_threshold=0** and **allocation_threshold=0**.

minimum – dumps only the allocated pages, which results in the smallest possible archive image. This option is useful when dumping to archive devices for which the throughput is much smaller than that of the database devices such as QIC tape drives. Using this option has the same effect as using the options **reserved_threshold=100** and **allocation_threshold=100**.

default – specifies that default values should be used.

When used with **archive_space**, this option dumps the database with the **reserved_threshold** and **allocation_threshold** options set to their default values. Use this to reset Backup Server to the default configuration.

When used with **reserved_threshold**, **default** specifies 85 percent.

When used with **allocation_threshold**, **default** specifies 40 percent.

reserved_threshold – dumps all the pages belonging to the database in a database disk if the percentage of reserved pages in the disk is

equal to or greater than *nnn*. For example, if you specify *nnn* as **60** and if a database disk has a percentage of reserved pages equal to or greater than 60 percent, then the entire disk is dumped without determining which pages within that disk are allocated. The default for this option is 85 percent.

nnn – an integer value between 0 and 100 that represents the value of the threshold. It is used to determine how much data to dump.

When used with **reserved_threshold**, if the percentage of reserved pages in the disk is greater than the value specified, all the pages of the database in a database disk are dumped.

When used with **allocation_threshold**, if the percentage of allocated pages in an allocation unit is greater than the percentage specified for **allocation_threshold**, all the pages within an allocation unit are dumped.

allocation_threshold – dumps all the pages in the allocation unit if the percentage of allocated pages in the unit is equal to or greater than *nnn*. For example, if *nnn* is specified as **70** and if the percentage of allocated pages in an allocation unit is equal to or greater than 70 percent, then the entire allocation unit is dumped without determining whether pages within that allocation unit are allocated or not. If the **reserved_threshold** setting causes the whole disk to be dumped, the **allocation_threshold** setting is ignored for the disk. The default for this option is 40 percent.

Examples

1. sp_dumpoptimize 'archive_space=maximum'

```
Backup Server: 4.172.1.1: The value of ‘reserved pages threshold’ has
been set to 0%.

Backup Server: 4.172.1.2: The value of ‘allocated pages threshold’ has
been set to 0%.
```

This causes the whole database to be dumped.

2. sp_dumpoptimize 'archive_space=minimum'

```
Backup Server: 4.172.1.1: The value of ‘reserved pages threshold’ has
been set to 100%.

Backup Server: 4.172.1.2: The value of ‘allocated pages threshold’ has
been set to 100%.
```

This causes only the allocated pages to be dumped, thereby resulting in the smallest archive image.

3. sp_dumpoptimize 'archive_space=default'

```
Backup Server: 4.172.1.1: The value of 'reserved pages threshold' has
been set to 85%.

Backup Server: 4.172.1.2: The value of 'allocated pages threshold' has
been set to 40%.
```

This causes the reserved threshold to be set to 85 percent and the allocation threshold to be set to 40 percent.

4. **sp_dumpoptimize 'reserved_threshold=60'**

```
Backup Server: 4.172.1.3: The value of 'reserved pages threshold' has
been set to 60%.
```

Those disks in the database whose percentage of reserved pages is greater than or equal to 60 percent are dumped without reading allocation pages on this disk. For the remaining disks, the allocation pages are read, and the last set value for the **allocation_threshold** is used. If the **allocation_threshold** was not set after Backup Server was started, default **allocation_threshold** of 40 percent is used.

5. **sp_dumpoptimize 'reserved_threshold=default'**

```
Backup Server: 4.172.1.3: The value of 'reserved pages threshold' has
been set to 85%.
```

This causes the reserved threshold to be set to 85 percent. It does not affect the allocation page threshold.

6. **sp_dumpoptimize 'allocation_threshold=80'**

```
Backup Server: 4.172.1.4: The value of 'allocated pages threshold' has
been set to 80%.
```

Allocation pages are read for those disks whose reserved page percentage is less than the last set value for the **reserved_threshold** and if an allocation unit has 80 percent or more pages allocated, then the whole allocation unit is dumped.

7. **sp_dumpoptimize 'allocation_threshold=default'**

```
Backup Server: 4.172.1.4: The value of 'allocated pages threshold' has
been set to 40%.
```

This causes the allocation page threshold to be set to the default of 40 percent. It does not affect the reserved pages threshold.

8. **sp_dumpoptimize 'reserved_threshold=60', 'allocation_threshold=30'**

```
Backup Server: 4.172.1.3: The value of 'reserved pages threshold' has
been set to 60%.

Backup Server: 4.172.1.4: The value of 'allocated pages threshold' has
been set to 30%.
```

Those disks in the database whose percentage of reserved pages is greater than or equal to 60 percent are dumped without reading allocation pages on this disk. For the remaining disks, the allocation pages are read and if an allocation unit has 30 percent or more pages allocated, then the whole allocation unit is dumped.

9. sp_dumpoptimize

```
Backup Server: 4.171.1.1: The current value of 'reserved pages
threshold' is 60%.

Backup Server: 4.171.1.2: The current value of 'allocated pages
threshold' is 30%.
```

This displays the current value of the thresholds.

Comments

- When you set values with **sp_dumpoptimize**, those values are immediately in affect without the need to restart Backup Server. However, the changes are effective only until the Backup Server is restarted. When Backup Server is restarted, the default values are used.
- If you issue **sp_dumpoptimize** multiple times, the thresholds specified by the last instance are used by later dumps. For example, if you first set the **reserved_threshold** value, and later issue **archive_space=maximum**, then that value overwrites the previous value you set for **reserved_threshold**.
- Dumps of different databases can use different thresholds by changing the **sp_dumpoptimize** values before each database dump.
- The optimal threshold values can vary from one database to another. Therefore, the performance of a dump depends on both the I/O configuration and the amount of used space in the database. The DBA can determine the appropriate configuration for a database by experimenting with dumps using different values and choosing the one that results in the shortest dump time.
- You can use **sp_dumpoptimize** for both local and remote dumps.
- **sp_dumpoptimize** has no effect on the performance of a transaction log dump or a load. Therefore, it need not be issued before **dump transaction, load database** or **load transaction** operations.
- If **sp_dumpoptimize** is issued without any parameters, the current value of the thresholds is displayed on the client.

- On configurations in which the archive device throughput is equal to or higher than the cumulative throughput of all the database disks, using **archive_space=maximum** may result in a faster dump. However, on configurations in which the archive device throughput is less than the cumulative throughput of all the database disks, using this option may result in a slower dump.
- The option names and the values for this procedure can be abbreviated to the unique substring that identifies them. For example, **ar = ma** is sufficient to uniquely identify the option **archive_space=maximum**.
- There can be zero or more blank space characters around the equal sign (**=**) in the option string.
- The option names and their values are case insensitive.
- For information on allocation pages, see "Understanding Page and Object Allocation Concepts" in Chapter 25, "Checking Database Consistency" in the *System Administration Guide.*

Permissions

Only the System Administrator, the Database Owner, or users with the Operator role can execute **sp_dumpoptimize**.

See Also

Commands	**dump database, dump transaction, load database, load transaction**

sp_estspace

Function

Estimates the amount of space required for a table and its indexes, and the time needed to create the index.

Syntax

```
sp_estspace table_name, no_of_rows [, fill_factor
   [, cols_to_max [, textbin_len [, iosec]]]]
```

Parameters

table_name – is the name of the table. It must already exist in the current database.

no_of_rows – is the estimated number of rows that the table will contain.

fill_factor – is the index fillfactor. The default is null, which means that Adaptive Server uses its default fillfactor.

cols_to_max – is a comma-separated list of the variable-length columns for which you want to use the maximum length instead of the average. The default is the average declared length of the variable-length columns.

textbin_len – is the length, per row, of all *text* and *image* columns. The default value is 0. You need to provide a value only if the table stores *text* or *image* data. *text* and *image* columns are stored in a separate set of data pages from the rest of the table's data. The actual table row stores a pointer to the *text* or *image* value. **sp_estspace** provides a separate line of information about the size of the *text* or *image* pages for a row.

iosec – is the number of disk I/Os per second on this machine. The default is 30 I/Os per second.

Examples

1. sp_estspace titles, 10000, 50, "title,notes", 0, 25

```
name             type         idx_level Pages        Kbytes
---------------- ------------ --------- ------------ ------------
titles           data                 0         3364         6728
titles           text/image           0            0            0
titleidind       clustered            0           21           43
titleidind       clustered            1            1            2
titleind         nonclustered         0         1001         2002
titleind         nonclustered         1           54          107
titleind         nonclustered         2            4            8
titleind         nonclustered         3            1            2

Total_Mbytes
-----------------
             8.68

name                 type         total_pages  time_mins
-------------------- ------------ ------------ ------------
titleidind           clustered            3386           13
titleind             nonclustered         1060            5
titles               data                    0            2
```

Calculates the space requirements for the *titles* table and its indexes, and the time required to create the indexes. The number of rows is 10,000, the fillfactor is 50 percent, two variable-length columns are computed using the maximum size for the column, and the disk I/O speed is 25 I/Os per second.

2. declare @i int
select @i = avg(datalength(pic)) from au_pix
exec sp_estspace au_pix, 1000, null, null, 16, @i

```
au_pix has no indexes
 name             type         idx_level Pages     Kbytes
 ---------------- ------------ --------- --------- ---------
 au_pix           data                 0        31        63
 au_pix           text/image           0     21000     42000

 Total_Mbytes
 -----------------
             41.08
```

Uses the average length of existing *image* data in the *au_pix* table to calculate the size of the table with 1000 rows. You can also provide this size as a constant.

3. sp_estspace titles, 50000

```
name             type         idx_level Pages        Kbytes
---------------- ------------ --------- ------------ ------------
titles           data                 0         4912         9824
titleidind       clustered            0           31           61
titleidind       clustered            1            1            2
titleind         nonclustered         0         1390         2780
titleind         nonclustered         1           42           84
titleind         nonclustered         2            2            4
titleind         nonclustered         3            1            2

Total_Mbytes
-----------------
            12.46

name               type         total_pages  time_mins
------------------ ------------ ------------ ------------
titleidind         clustered            4943           19
titleind           nonclustered         1435            8
```

Calculates the size of the *titles* table with 50,000 rows, using defaults for all other values.

Comments

- To estimate the amount of space required by a table and its indexes:

  a. Create the table.

  b. Create all indexes on the table.

  c. Run **sp_estspace**, giving the table name, the estimated number of rows for the table, and the optional arguments, as needed.

  You do not need to insert data into the tables. **sp_estspace** uses information in the system tables—not the size of the data in the tables—to calculate the size of tables and indexes.

- If the **auto identity** option is set in a database, Adaptive Server automatically defines a 10-digit IDENTITY column in each new table that is created without specifying a **primary** key, a **unique** constraint, or an IDENTITY column. To estimate how much extra space is required by this column:

  a. In the master database, use **sp_dboption** to turn on the **auto identity** option for the database.

  b. Create the table.

  c. Run **sp_estspace** on the table and record the results.

d. Drop the table.

e. Turn the **auto identity** option off for the database.

f. Re-create the table.

g. Rerun **sp_estspace** on the table, and record the results.

- For information about tables or columns, use **sp_help** *tablename*.

Permissions

Any user can execute **sp_estspace**.

Tables Used

syscolumns, *sysindexes*, *sysobjects*

See Also

Commands	**create index**, **create table**
System procedures	**sp_help**

sp_export_qpgroup

Function

Exports all plans for a specified user and abstract plan group to a user table.

Syntax

sp_export_qpgroup *usr*, *group*, *tab*

Parameters

usr – is the name of the user who owns the abstract plans to be exported.

group – is the name of the abstract plan group that contains the plans to be exported.

tab – is the name of a table into which to copy the plans. It must be a table in the current database. You can specify a database name, but not an owner name, in the form *dbname..tablename*. The total length must be 30 characters or less.

Examples

1. **sp_export_qpgroup freidak, ap_stdout, "tempdb..moveplans"**

   Creates a table called *moveplans* containing all the plans for the user "freidak" that are in the *ap_stdout* group.

Comments

- **sp_export_qpgroup** copies plans from an abstract plan group to a user table. With **sp_import_qpgroup**, it can be used to copy abstract plans groups between servers and databases or to assign user IDs to copied plans.
- The user table name that you specify cannot exist before you run **sp_export_qpgroup**. The table is created with a structure identical to that of *sysqueryplans*.
- **sp_export_qpgroup** uses **select...into** to create the table to store the copied plans. You must use **sp_dboption** to enable **select into/bulkcopy/pllsort** in order to use **sp_export_qpgroup**, or create the table in *tempdb*.

Permissions

Only a System Administrator or the Database Owner can execute **sp_export_qpgroup.**

Tables Used

sysattributes, sysqueryplans

See Also

System procedures	**sp_copy_all_qplans**, **sp_copy_qplan**, **sp_import_qpgroup**

sp_extendsegment

Function

Extends the range of a segment to another database device.

Syntax

```
sp_extendsegment segname, dbname, devname
```

Parameters

segname – is the name of the existing segment previously defined with **sp_addsegment**.

dbname – is the name of the database on which to extend the segment. *dbname* must be the name of the current database.

devname – is the name of the database device to be added to the current database device range already included in *segname*.

Examples

```
1. sp_extendsegment indexes, pubs2, dev2
```

This command extends the range of the segment *indexes* for the database *pubs2* on the database device *dev2*.

Comments

- After defining a segment, you can use it in the **create table** and **create index** commands to place the table or index on the segment. If you create a table or index on a particular segment, subsequent data for the table or index is located on that segment.
- To associate a segment with a database device, create or alter the database with a reference to that device. A database device can have more than one segment associated with it.
- A segment can be extended over several database devices.
- When you extend the *logsegment* segment, Adaptive Server recalculates its last-chance threshold.

Permissions

Only the Database Owner or a System Administrator can execute **sp_extendsegment**.

Tables Used

master.dbo.sysdatabases, sysdevices, master.dbo.sysusages, sysobjects, syssegments

See Also

Commands	**alter database**, **create index**, **create table**
System procedures	**sp_addsegment**, **sp_dropsegment**, **sp_helpdb**, **sp_helpdevice**, **sp_helpsegment**, **sp_placeobject**

sp_familylock

Function

Reports information about all the locks held by a family (coordinating process and its worker processes) executing a statement in parallel.

Syntax

sp_familylock [*fpid1* [, *fpid2*]]

Parameters

fpid1 – is the family identifier for a family of worker processes from the *master.dbo.sysprocesses* table. Run **sp_who** or **sp_lock** to get the *spid* of the parent process.

pfid2 – is the Adaptive Server process ID number for another lock.

Examples

1. sp_familylock 5

```
fid  spid locktype     table_id   page dbname  class            context
---  ---- ----------   --------   ---- ------  ---------------  --------
5    5    Sh_intent     176003658  0    userdb  Non cursor lock Sync-
pt duration request
5    5    Sh_intent-blk 208003772  0    userdb  Non cursor lock Sync-
pt duration request
5    6    Sh_page       208003772  3972 userdb  Non cursor lock Sync-
pt duration request
5    7    Sh_page       208003772  3973 userdb  Non cursor lock Sync-
pt duration request
5    8    Sh_page       208003772  3973 userdb  Non cursor lock Sync-
pt duration request
```

Displays information about the locks held by all members of the family with an *fid* of 5.

Comments

- **sp_familylock** with no parameter reports information on all processes belonging to families that currently hold locks. The report is identical to the output from **sp_lock**; however, **sp_familylock** allows you to generate reports based on the family ID, rather than the process ID. It is useful for detecting family deadlocks.

- Use the **object_name** system function to derive a table's name from its ID number.
- The "locktype" column indicates whether the lock is a shared lock ("Sh" prefix), an exclusive lock ("Ex" prefix) or an update lock, and whether the lock is held on a table ("table" or "intent") or on a page ("page").

  The "blk" suffix in the "locktype" column indicates that this process is blocking another process that needs to acquire a lock. As soon as this process completes, the other process(es) moves forward. The "demand" suffix indicates that the process is attempting to acquire an exclusive lock.
- The "class" column indicates whether a lock is associated with a cursor. It displays one of the following:
  - "Non cursor lock" indicates that the lock is not associated with a cursor.
  - "Cursor Id *number*" indicates that the lock is associated with the cursor ID number for that Adaptive Server process ID.
  - A cursor name indicates that the lock is associated with the cursor *cursor_name* that is owned by the current user executing **sp_lock**.
- The "fid" column identifies the family (including the coordinating process and its worker processes) to which a lock belongs. Values for "fid" are as follows:
  - A zero value indicates that the task represented by the *spid* is executed in serial. It is not participating in parallel execution.
  - A nonzero value indicates that the task (*spid*) holding the lock is a member of a family of processes (identified by "fid") executing a statement in parallel. If the value is equal to the *spid*, it indicates that the task is the coordinating process in a family executing a query in parallel.
- The "context" column identifies the context of the lock. Worker processes in the same family have the same context value. Values for "context" are as follows:
  - "NULL" means that the task holding this lock is either executing a query in serial or is a query being executed in parallel in transaction isolation level 1.
  - "FAM_DUR" means that the task holding the lock will hold the lock until the query is complete.

A lock's context may be "FAM_DUR" if the lock is a table lock held as part of a parallel query, if the lock is held by a worker process at transaction isolation level 3, or if the lock is held by a worker process in a parallel query and must be held for the duration of the transaction.

Permissions

Any user can execute **sp_familylock**.

Tables Used

master.dbo.spt_values, *master.dbo.syslocks*, *sysobjects*, *master.dbo.sysprocesses*.

See Also

Commands	**kill**, **select**
System procedures	**sp_lock**, **sp_who**

sp_find_qplan

Function

Finds an abstract plan, given a pattern from the query text or plan text.

Syntax

sp_find_qplan *pattern* [, group]

Parameters

pattern – is a string to find in the text of the query or abstract plan.

group – is the name of the abstract plan group.

Examples

1. sp_find_qplan "%from titles%"

```
gid         id
          text
----------- -----------
         ------------------------------------------------------
           2 921054317
          select count(*) from titles
           2 921054317
          ( plan
         ( i_scan t_pub_id_ix titles )
         ( )
)
( prop titles
         ( parallel 1 )
         ( prefetch 16 )
         ( lru )
)
           5   937054374
          select type, avg(price) from titles group by type
           5   937054374
          ( plan
         ( store Worktab1
                ( i_scan type_price titles )
         )
         ( t_scan ( work_t Worktab1 ) )
)
( prop titles
         ( parallel 1 )
         ( prefetch 16 )
         ( lru )
```

Reports on all abstract plans that have the string “from titles” in the query.

2. `sp_find_qplan "%t_scan%"`

   Finds all plans that include a table scan operator.

3. `sp_find_qplan "%table[0-9]%", dev_plans`

   Uses the range pattern matching to look for strings such as “table1”, “table2”, and so forth, in plans in the *dev_plans* group.

Comments

- Use **sp_find_qplan** to find an abstract plan that contains a particular string. You can match strings from either the query text or from the abstract plan text.
- For each matching plan, **sp_find_qplan** prints the group ID, plan ID, query text and abstract plan text.
- If you include a group name, **sp_find_qplan** searches for the string in the specified group. If you do not provide a group name, **sp_find_plan** searches all queries and plans for all groups.
- You must supply the “%” wildcard characters, as shown in the examples, unless you are searching for a string at the start or end of a query or plan. You can use any Transact-SQL pattern matching syntax, such as that shown in Example 3.
- The text of queries in *sysqueryplans* is broken into 255-byte column values. **sp_find_qplan** may miss matches that span one of these boundaries, but finds all matches that are less than 127 bytes, even if they span two rows.

Permissions

Any user can execute **sp_find_qplan**. It reports only on abstract plans owned by the user who executes it, except when executed by a System Administrator or the Database Owner.

Tables Used

sysattributes, *sysqueryplans*

See Also

System procedures	**sp_help_qpgroup**, **sp_help_qplan**

sp_flushstats

Function

Flushes statistics from in-memory storage to the *systabstats* system table.

Syntax

```
sp_flushstats objname
```

Parameters

objname – is the name of a table.

Examples

```
1. sp_flushstats titles
```

Flushes statistics for the *titles* table.

Comments

- Some statistics in the *systabstats* table are updated in in-memory storage locations and flushed to *systabstats* periodically, to reduce overhead and contention on *systabstats*.
- If you query *systabstats* using SQL, executing **sp_flushstats** guarantees that in-memory statistics are flushed to *systabstats*.
- The **optdiag** command always flushes in-memory statistics before displaying output.
- The statistics in *sysstatistics* are changed only by data definition language commands and do not require the use of **sp_flushstats**.

Permissions

Only a System Administrator can execute **sp_flushstats**.

sp_forceonline_db

Function

Provides access to all the pages in a database that were previously marked suspect by recovery.

Syntax

```
sp_forceonline_db dbname,
   {"sa_on" | "sa_off" | "all_users"}
```

Parameters

dbname – is the name of the database to be brought online.

sa_on – allows only users with the **sa_role** access to the specified page.

sa_off – revokes access privileges created by a previous invocation of **sp_forceonline_page** with **sa_on**.

all users – allows all users access to the specified page.

Examples

1. `sp_forceonline_db pubs2, "sa_on"`

   Allows the System Administrator access to all suspect pages in the *pubs2* database.

2. `sp_forceonline_db pubs2, "sa_off"`

   Revokes access to all suspect pages in the *pubs2* database from the System Administrator. Now, no one can access the suspect pages in *pubs2*.

3. `sp_forceonline_db pubs2, "all_users"`

   Allows all users access to all pages in the *pubs2* database.

Comments

- A page that is forced online is not necessarily repaired. Corrupt pages can also be forced online. Adaptive Server does not perform any consistency checks on pages that are forced online.
- **sp_forceonline_page** with **all users** cannot be reversed. When pages have been brought online for all users, you cannot take them offline again.
- **sp_forceonline_db** cannot be used in a transaction.

- To bring only specific offline pages online, use **sp_forceonline_page.**

Permissions

Only a System Administrator can execute **sp_forceonline_db.**

Tables Used

master.dbo.sysattributes

See Also

System procedures	**sp_forceonline_page, sp_listsuspect_db**, **sp_listsuspect_page**, **sp_setsuspect_granularity**, **sp_setsuspect_threshold**

sp_forceonline_object

Function

Provides access to an index previously marked suspect by recovery.

Syntax

```
sp_forceonline_object dbname, objname, indid,
   {sa_on | sa_off | all_users} [, no_print]
```

Parameters

dbname – is the name of the database containing the index to be brought online.

objname – is the name of the table.

indid - is the index ID of the suspect index being brought online.

sa_on – allows only users with the **sa_role** to access the specified index.

sa_off – revokes access privileges created by a previous invocation of **sp_forceonline_object** with **sa_on.**

all_users – allows all users to access the specified index.

no_print - skips printing a list of other suspect objects after the specified object is brought online.

Examples

1. **`sp_forceonline_object pubs2, titles, 3 , sa_on`**

   Allows a System Administrator to access the index with indid 3 on the *titles* table in the *pubs2* database.

2. **`sp_forceonline_object pubs2, titles 3, sa_off`**

   Revokes access to the index from the System Administrator. Now, no one has access to this index.

3. **`sp_forceonline_object pubs2, titles, 3, all_users`**

   Allows all users to access the index on the *titles* table in the *pubs2* database.

Comments

- If an index on a data-only-locked table has suspect pages, the entire index is taken offline during recovery. Offline indexes are

not considered by the query optimizer. Indexes on allpages-locked tables are not taken completely offline during recovery; only individual pages of these indexes are taken offline. These pages can be brought online with **sp_forceonline_page**.

- Use **sp_listsuspect_object** to see a list of databases that are offline.
- To repair a suspect index, use **sp_forceonline_object** with **sa_on** access. Then, drop and re-create the index.

➤ ***Note***

If the index is on *systabstats* or *sysstatistics* (the only data-only-locked system tables) call Sybase Technical Support for assistance.

- **sp_forceonline_object** with **all_users** cannot be reversed. When an index has been brought online for all users, you cannot take it offline again.
- An index that is forced online is not necessarily repaired. Corrupt indexes can be forced online. Adaptive Server does not perform any consistency checks on indexes that are forced online.
- **sp_forceonline_object** cannot be used in a transaction.
- **sp_forceonline_object** works only for databases in which the recovery fault isolation mode is "page." Use **sp_setsuspect_granularity** to display the recovery fault isolation mode for a database.
- To bring all of a database's offline pages and indexes online in a single command, use **sp_forceonline_db**.
- For more information on recovery fault isolation, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_forceonline_object**.

Tables Used

master.dbo.sysattributes

See Also

System procedures	**sp_listsuspect_object**

sp_forceonline_page

Function

Provides access to pages previously marked suspect by recovery.

Syntax

```
sp_forceonline_page dbname, pgid,
   {"sa_on" | "sa_off" | "all_users"}
```

Parameters

dbname – is the name of the database containing the pages to be brought online.

pgid – is the page identifier of the page being brought online.

sa_on – allows only users with the **sa_role** access to the specified page.

sa_off – revokes access privileges created by a previous invocation of **sp_forceonline_page** with **sa_on**.

all_users – allows all users access to the specified page.

Examples

1. `sp_forceonline_page pubs2, 312, "sa_on"`

   Allows a System Administrator access to page 312 in the *pubs2* database.

2. `sp_forceonline_page pubs2, 312, "sa_off"`

   Revokes access to page 312 in the *pubs2* database from the System Administrator. Now, no one has access to this page.

3. `sp_forceonline_page pubs2, 312, "all_users"`

   Allows all users access to page 312 in the *pubs2* database.

Comments

- **sp_forceonline_page** with **all_users** cannot be reversed. When pages have been brought online for all users, you cannot take them offline again.
- A page that is forced online is not necessarily repaired. Corrupt pages can also be forced online. Adaptive Server does not perform any consistency checks on pages that are forced online.
- **sp_forceonline_page** cannot be used in a transaction.

- **sp_forceonline_page** works only for databases in which the recovery fault isolation mode is "page." Use **sp_setsis[ect_granularity** to display the recovery fault isolation mode for a database.
- To bring all of a database's offline pages online in a single command, use **sp_forceonline_db**.

Permissions

Only a System Administrator can use **sp_forceonline_page.**

Tables Used

master.dbo.sysattributes

See Also

System procedures	**sp_forceonline_db, sp_listsuspect_db**, **sp_listsuspect_page**, **sp_setsuspect_granularity**, **sp_setsuspect_threshold**

sp_foreignkey

Function

Defines a foreign key on a table or view in the current database.

Syntax

```
sp_foreignkey tabname, pktabname, col1 [, col2] ...
   [, col8]
```

Parameters

tabname – is the name of the table or view that contains the foreign key to be defined.

pktabname – is the name of the table or view that has the primary key to which the foreign key applies. The primary key must already be defined.

col1 – is the name of the first column that makes up the foreign key. The foreign key must have at least one column and can have a maximum of eight columns.

Examples

1. `sp_foreignkey titles, publishers, pub_id`

   The primary key of the *publishers* table is the *pub_id* column. The *titles* table also contains a *pub_id* column, which is a foreign key of *publishers*.

2. `sp_foreignkey orders, parts, part, subpart`

   The primary key of the *parts* table has been defined with **sp_primarykey** as the *partnumber* and *subpartnumber* columns. The *orders* table contains the columns *part* and *subpart*, which make up a foreign key of *parts*.

Comments

- **sp_foreignkey** adds the key to the *syskeys* table. Keys make explicit a logical relationship that is implicit in your database design.
- **sp_foreignkey** does not enforce referential integrity constraints; use the **foreign key** clause of the **create table** or **alter table** command to enforce a foreign key relationship.
- The number and order of columns that make up the foreign key must be the same as the number and order of columns that make

up the primary key. The datatypes (and lengths) of the primary and foreign keys must agree, but the null types need not agree.

- The installation process runs **sp_foreignkey** on the appropriate columns of the system tables.
- To display a report on the keys that have been defined, execute **sp_helpkey.**

Permissions

Only the owner of the table or view can execute **sp_foreignkey.**

Tables Used

syscolumns, sysindexes, syskeys, sysobjects, sysreferences

See Also

Commands	**alter table**, **create table**, **create trigger**
System procedures	**sp_commonkey**, **sp_dropkey**, **sp_helpkey**, **sp_helpjoins**, **sp_primarykey**

sp_freedll

Function

Unloads a dynamic link library (DLL) that was previously loaded into XP Server memory to support the execution of an extended stored procedure (ESP).

Syntax

```
sp_freedll dll_name
```

Parameters

dll_name – is the file name of the DLL being unloaded from XP Server memory.

Examples

```
1. sp_freedll "sqlsrvdll.dll"
```

Unloads the *sqlsrvdll.dll* DLL.

Comments

- **sp_freedll** cannot be executed from within a transaction.
- **sp_freedll** cannot free the DLL of a system ESP.
- An alternative to unloading a DLL explicitly, using **sp_freedll**, is to specify that DLLs always be unloaded after the ESP request that invoked them terminates. To do this, set the **esp unload dll** configuration parameter to 1 or start **xpserver** with the **-u** option.
- **sp_freedll** can be used to update an ESP function in a DLL without shutting down XP Server or Adaptive Server.
- If you use **sp_freedll** to unload a DLL that is in use, **sp_freedll** will succeed, causing the ESP currently using the DLL to fail.

Permissions

Only a System Administrator can execute **sp_freedll**.

Tables Used

master.dbo.syscomments, *sysobjects*

See Also

System procedures	**sp_addextendedproc, sp_dropextendedproc, sp_helpextendedproc**
Utility	

sp_getmessage

Function

Retrieves stored message strings from *sysmessages* and *sysusermessages* for **print** and **raiserror** statements.

Syntax

```
sp_getmessage message_num, result output [, language]
```

Parameters

message_num – is the number of the message to be retrieved.

result **output** – is the variable that receives the returned message text, followed by a space and the keyword **output**. The variable must have a datatype of *char*, *nchar*, *varchar*, or *nvarchar*.

language – is the language of the message to be retrieved. *language* must be a valid language name in *syslanguages* table. If you include *language*, the message with the indicated *message_num* and *language* is retrieved. If you do not include *language*, then the message for the default session language, as indicated by the variable *@@langid*, is retrieved.

Examples

1.
```
declare @myvar varchar(200)
exec sp_getmessage 20001, @myvar output
```

   Retrieves message number 20001 from *sysusermessages*.

2.
```
declare @myvar varchar(200)
exec sp_getmessage 20010, @myvar output, french
```

   Retrieves the French language version of message number 20010 from *sysusermessages*.

Comments

- Any application can use **sp_getmessage**, and any user can read the messages stored in *sysmessages* and *sysusermessages*.

Permissions

Any user can execute **sp_getmessage**.

Tables Used

master.dbo.syslanguages, *master.dbo.sysmessages*, *sysobjects*, *sysusermessages*

See Also

Commands	**print**, **raiserror**
System procedures	**sp_addmessage**, **sp_dropmessage**

sp_grantlogin

(Windows NT only)

Function

Assigns Adaptive Server roles or **default** permissions to Windows NT users and groups when Integrated Security mode or Mixed mode (with Named Pipes) is active.

Syntax

```
sp_grantlogin {login_name | group_name}
   ["role_list" | default]
```

Parameters

login_name – is the network login name of the Windows NT user.

group_name – is the Windows NT group name.

role_list – is a list of the Adaptive Server roles granted. The role list can include one or more of the following role names: **sa_role**, **sso_role**, **oper_role**. If you specify more than one role, separate the role names with spaces, not commas.

default – specifies that the *login_name* or *group_name* receive default permissions assigned with the **grant** statement or **sp_role** procedure.

Examples

1. `sp_grantlogin jeanluc, oper_role`

   Assigns the Adaptive Server **oper_role** to the Windows NT user "jeanluc".

2. `sp_grantlogin valle`

   Assigns the **default** value to the Windows NT user "valle". User "valle" receives any permissions that were assigned to her via the **grant** command or **sp_role** procedure.

3. `sp_grantlogin Administrators, "sa_role sso_role"`

   Assigns the Adaptive Server **sa_role** and **sso_role** to all members of the Windows NT administrators group.

Comments

- You must create the Windows NT login name or group before assigning roles with **sp_grantlogin**. See your Windows NT documentation for details.
- **sp_grantlogin** is active only when Adaptive Server is running in Integrated Security mode or Mixed mode when the connection is Named Pipes. If Adaptive Server is running under Standard mode or Mixed mode with a connection other than Named Pipes, use **grant** and **sp_role** instead.
- If you do not specify a *role_list* or **default**, the procedure automatically assigns the **default** value.
- The **default** value does not indicate an Adaptive Server role. It specifies that the user or group should receive any permissions that were assigned to it via the **grant** command or **sp_role** procedure.
- Using **sp_grantlogin** with an existing *login_name* or *group_name* overwrites the user's or group's existing roles.

Permissions

Only a System Administrator can execute **sp_grantlogin**.

Tables Used

sysobjects

See Also

Commands	**grant**, **setuser**
System procedures	**sp_addlogin**, **sp_addremotelogin**, **sp_adduser**, **sp_displaylogin**, **sp_droplogin**, **sp_dropuser**, **sp_locklogin**, **sp_logininfo**, **sp_modifylogin**, **sp_revokelogin**, **sp_who**

sp_ha_admin

Function

Performs administrative tasks on Adaptive Servers configured with Sybase Failover in a high availability system. **sp_ha_admin** is installed with the *installhavss* script on UNIX platforms or the *insthasv* script on Windows NT.

Syntax

```
sp_ha_admnin [cleansessions | help]
```

Parameters

cleansessions – Removes old entries from *syssessions*. Old *syssessions* entries are typically left behind because either Adaptive Server failed to clean up *syssessions* during a reboot, or because a client failed to connect to Adaptive Server.

help – displays the syntax for **sp_ha_admin**.

Examples

1. **sp_ha_admin cleansessions**
   (return status = 0)

   Removes old entries from *syssessions* left by a client connection that did not exit correctly.

2. **sp_ha_admin "help"**

   ```
   sp_ha_admin Usage: sp_ha_admin command [, option1
   [, option2]]
   sp_ha_admin commands:
   sp_ha_admin 'cleansessions'
   sp_ha_admin 'help'
   (return status = 0)
   ```

   Displays the syntax for **sp_ha_admin**

Comments

- **sp_ha_admin** performs administrative tasks on Adaptive Server that are configured for Sybase's Failover in a high availability system. **sp_ha_admin** is not installed using the *installmaster* script; instead, use the *installhavss* script that installs and configures for Sybase's Failover (*insthasv* on Windows NT).
- **sp_ha_admin** returns a 0 if it successfully cleaned up *syssessions*, and returns a 1 if it encounters an error.

- **sp_ha_admin** enters a message in the errorlog if it could not remove any entries from *syssessions* (for example, if it could not get a lock on *syssessions*).
- To view all the current entries in *syssessions*, enter:

```
select * from syssessions
```

Permissions

Only the a System Administrator with the **ha_role** can execute **sp_ha_admin.**

Tables Used

master.dbo.syssesions

sp_help

Function

Reports information about a database object (any object listed in *sysobjects*) and about system or user-defined datatypes.

Syntax

sp_help [*objname*]

Parameters

objname – is the name of any object in *sysobjects* or any user-defined datatype or system datatype in *systypes*. You cannot specify database names. *objname* can include tables, views, stored procedures, logs, rules, defaults, triggers, referential constraints, and check constraints. Use owner names if the object owner is not the user running the command and is not the Database Owner.

Examples

1. sp_help

Displays a list of objects in *sysobjects* and displays each object's name, owner, and object type. Also displays a list of each user-defined datatype in *systypes*, indicating the datatype name, storage type, length, null type, default name, and rule name. Null type is 0 (null values not allowed) or 1 (null values allowed).

2. sp_help publishers

```
Name                        Owner                          Type
--------------------------- -----------------------------  -----------
publishers                  dbo                            user table

Data_located_on_segment        When_created
------------------------------ -------------------------
default                                 Apr 12 1999  3:31PM

Column_name Type Length Prec Scale Nulls Default_name Rule_name Identity
----------- ---- ------ ---- ----- ----- ------------ --------- --------
pub_id      char      4 NULL  NULL     0 NULL         pub_idrule       0
pub_name varchar     40 NULL  NULL     1 NULL         NULL             0
city     varchar     20 NULL  NULL     1 NULL         NULL             0
state       char      2 NULL  NULL     1 NULL         NULL             0

attribute_class attribute     int_value char_value            comments
--------------- ------------- --------- --------------------- --------
buffer manager  cache binding         1 publishers_cache          NULL
```

```
index_name         index_description                      index_keys
     index_max_rows_per_page
------------------ -------------------------------------- ----------
     ------------------------
pubind             clustered, unique located on default   pub_id
     0
name    attribute_class attribute  int_value char_value
         comments
------- --------------- ---------- --------- --------------------------
         --------
pubind  buffer manager  cache name      NULL cache for index pubind
         NULL
keytype     object                    related_object
         object_keys
         related_keys
----------- ------------------------ ----------------------------
        ------------
        ------------
primary     publishers                - none --
        pub_id, *, *, *, *, *, *, *
        *, *, *, *, *, *, *, *
foreign     titles                         publishers
        pub_id, *, *, *, *, *, *, *
        pub_id, *, *, *, *, *, *, *
Object is not partitioned.
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with allpages
lock scheme.
The attribute 'concurrency_opt_threshold' is not applicable to tables
with allpages lock scheme.

 exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ----------------- ------------
            0              0          0                 0            0
 concurrency_opt_threshold
 -------------------------
                         0
```

Displays information about the *publishers* table. **sp_help** also lists any attributes assigned to the specified table and its indexes, giving the attribute's class, name, integer value, character value, and comments. The above example shows cache binding attributes for the *publishers* table.

3. sp_help partitioned_table

```
Name                          Owner        Type
-------------------------     ------------ -----------
partitioned_table             dbo          user table

Data_located_on_segment       When_created
-----------------------       -------------------------
data1                         Mar 24 1995 10:48AM

Column_name  Type  Length Prec  Scale Nulls Default_name
-----------  ----- ------ ----  ----- ----- ------------
col1         char      5   NULL  NULL     0  NULL

Rule_name       Identity
----------      --------
NULL                   0

Object does not have any indexes.
No defined keys for this object.
partitionid      firstpage    controlpage
---------------- ------------ -----------
              1          145         146
              2          312         313
Lock scheme Datarows

 exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ----------------- ------------
            1              0          0                 0            0
 concurrency_opt_threshold
 -------------------------
                        15
```

Displays information about a partitioned table.

4. sp_help "mary.marytrig"

```
Name          Owner               Type
------------ ------------------- ----------------
marytrig      mary                trigger

Data_located_on_segment When_created
----------------------- --------------------------
not applicable          Mar 20 1992  2:03PM
```

Displays information about the trigger *marytrig* owned by user "mary". The quotes are needed, because the period is a special character.

5. sp_help money

```
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----

money      money             8  NULL  NULL
```

```
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    1   NULL          NULL                0
```

Displays information about the system datatype *money*.

6. sp_help identype

```
Type_name  Storage_type Length Nulls Default_name
--------- ------------ ------ ----- ----------
identype   numeric           4     0 NULL

Rule_name       Identity
---------       --------
NULL                   1
```

Displays information about the user-defined datatype *identype*. The report indicates the base type from which the datatype was created, whether it allows nulls, the names of any rules and defaults bound to the datatype, and whether it has the IDENTITY property.

7. sp_help titles

```
Name         Owner                   Type
------------ ----------------------- ----------------------
titles       dbo                     user table

Data_located_on_segment        When_created
------------------------------ --------------------------
default                         Dec  6 1997 12:07PM

Column_name     Type    Length Prec Scale Nulls Default_name Rule_name
Identity
--------------- ------- ------ ---- ----- ----- ------------ --------- -
-------
title_id        tid          6 NULL  NULL     0 NULL
NULL             0
title           varchar     80 NULL  NULL     0 NULL
NULL             0
type            char        12 NULL  NULL     0 typedflt
NULL             0
pub_id          char         4 NULL  NULL     1 NULL
NULL             0
price           money        8 NULL  NULL     1 NULL
NULL             0
advance         money        8 NULL  NULL     1 NULL
NULL             0
total_sales     int          4 NULL  NULL     1 NULL
NULL             0
notes           varchar    200 NULL  NULL     1 NULL
NULL             0
pubdate         datetime     8 NULL  NULL     0 datedflt
```

```
NULL             0
contract        bit          1 NULL  NULL     0 NULL
NULL             0

attribute_class        attribute              int_value
         char_value
         comments
------------------------------ ------------------------------ ----------
lock strategy                   row lock promotion
NULL
         PCT = 95, LWM = 300, HWM = 300
         NULL

index_name           index_description
        index_keys
        index_max_rows_per_page index_fillfactor index_reservepagegap
------------------- ------------------------------------------------------
-----
       ---------------------- ---------------- --------------------
titleidind           clustered, unique located on default
         title_id
                             0                0                    0
titleind             nonclustered located on default
         title
                             0                0                    0
type_price           nonclustered located on default
        type, price DESC
                           0               0                   0
No defined keys for this object.
Object is not partitioned.
Lock scheme Datarows

exp_row_size reservedpagegap fillfactor max_rows_per_page identity_gap
------------ -------------- ---------- ----------------- ------------
         224             16          0                 0            0
 concurrency_opt_threshold
 -------------------------
                         0
```

Reports on the *titles* table, including information about the locking scheme, expected row size, reserve page gap, and row lock promotion settings.

Comments

- **sp_help** looks for an object in the current database only.
- **sp_help** follows the Adaptive Server rules for finding objects:
  - If you do not specify an owner name, and you own an object with the specified name, **sp_help** reports on that object.

  - If you do not specify an owner name, and do not own an object of that name, but the Database Owner does, **sp_help** reports on the Database Owner's object.
  - If neither you nor the Database Owner owns an object with the specified name, **sp_help** reports an error condition, even if an object with that name exists in the database for a different owner. Qualify objects that are owned by database users other than yourself and the Database Owner with the owner's name, as shown in example 4.
  - If both you and the Database Owner own objects with the specified name, and you want to access the Database Owner's object, specify the name in the format *dbo.objectname*.
- **sp_help** works on temporary tables if you issue it from *tempdb*.
- Columns with the IDENTITY property have an "Identity" value of 1; others have an "Identity" value of 0. In example 2, there are no IDENTITY columns.
- **sp_help** lists any indexes on a table, including indexes created by defining unique or primary key constraints in the **create table** or **alter table** statements. It also lists any attributes associated with those indexes. However, **sp_help** does not describe any information about the integrity constraints defined for a table. Use **sp_helpconstraint** for information about any integrity constraints.
- **sp_help** displays the following new settings:
  - The locking scheme, which can be set with **create table** and changed with **alter table**
  - The expected row size, which can be set with **create table** and changed with **sp_chgattribute**
  - The reserve page gap, which can be set with **create table** and changed with **sp_chgattribute**
  - The row lock promotion settings, which can be set or changed with **sp_setrowlockpromote** and dropped with **sp_droprowlockpromote**
- **sp_help** includes the report from **sp_helpindex**, which shows the order of the keys used to create the index and the space management properties.
- When Component Integration Services is enabled, **sp_help** displays information on the storage location of remote objects.

Permissions

Any user can execute **sp_help.**

Tables Used

master.dbo.spt_values, master.sysattributes, sysattributes, syscolumns, sysindexes, sysmessages, sysobjects, syspartitions, systypes

See Also

System procedures	**sp_chgattribute**, **sp_droprowlockpromote**, **sp_helpartition**, **sp_helpconstraint**, **sp_helpdb**, **sp_helpindex**, **sp_helpkey**, **sp_helprotect**, **sp_helpsegment**, **sp_setrowlockpromote**, **sp_helptext**, **sp_helpuser**

sp_helpartition

Function

Lists the partition number, first page, control page, and number of data pages and summary size information for each partition in a partitioned table.

Syntax

sp_helpartition [*table_name*]

Parameters

table_name – is the name of a partitioned table in the current database. If the table name is not supplied, the owner, tables name, and number of partitions is printed for all user tables in the database.

Examples

1. sp_helpartition sales

```
partitionid firstpage   controlpage ptn_data_pages
----------- ----------- ----------- --------------
          1         313         314           4227
          2       12802       12801           4285
          3       25602       25601           4404
          4       38402       38401           4523
          5       51202       51201           4347
          6       64002       64001           4285

(6 rows affected)
Partitions Average Pages Maximum Pages Minimum Pages Ratio (Max/Avg)
---------- ------------- ------------- ------------- ---------------
         6           4345          4523          4227        1.040967
```

Returns information about the partitions in *sales*.

Comments

- **sp_helpartition** lists the partition number, first page, control page, and number of data pages for each partition in a partitioned table. The number of pages per partition shows how evenly the data is distributed between partitions.

  The summary information display the number of partitions, the average number of pages per partition, the minimum and maximum number of pages, and the ratio between the average number of pages and the maximum number. This ratio is used

during query optimization. If the ratio is 2 or greater (meaning that the maximum size is twice as large as the average size), the optimizer chooses a serial query plan rather than a parallel plan.

- Partitioning a table creates additional page chains. Use the partition clause of the **alter table** command to partition a table. Each chain has its own last page, which is available for concurrent insert operations. This improves insert performance by reducing page contention. If the table is spread over multiple physical devices, partitioning improves insert performance by reducing I/O contention while Adaptive Server is flushing data from cache to disk.
- Partitioning a table does not affect its performance for update or delete commands.
- Use the **unpartition** clause of the **alter table** command to concatenate all existing page chains.
- Neither partitioning nor unpartitioning a table moves existing data.
- To change the number of partitions in a table, first use the **unpartition** clause of **alter table** to concatenate its page chains. Then use the **partition** clause of **alter table** to repartition the table.
- **sp_helpartition** looks only in the current database for the table.
- Use **sp_helpsegment** to display the number of used and free pages on the segment on where the partitioned table is stored.

Accuracy of Results

- The values reported in the "data_pages" column may be greater than the actual values. To determine whether the count is inaccurate, run **sp_statistics** and **sp_helpartition** to compare the data page count. The count provided by **sp_statistics** is always accurate.

  If the page count reported by **sp_statistics** differs from the sum of the partition pages reported by **sp_helpartition** by more then 5 percent, run one of the following commands to update the partition statistics:

  - **dbcc checkalloc**
  - **dbcc checkdb**
  - **dbcc checktable**
  - **update all statistics**
  - **update partition statistics**

Then, rerun **sp_helpartition** for an accurate report.

Permissions

Any user can execute **sp_helpartition**.

Tables Used

syspartitions

See Also

Commands	**alter table**, **insert**
System procedures	**sp_help**, **sp_helpsegment**

sp_helpcache

Function

Displays information about the objects that are bound to a data cache or the amount of overhead required for a specified cache size.

Syntax

```
sp_helpcache {cache_name | "cache_size[P|K|M|G]"}
```

Parameters

cache_name – is the name of an existing data cache.

cache_size – specifies the size of the cache, specified by **P** for pages, **K** for kilobytes, **M** for megabytes, or **G** for gigabytes. The default is **K**.

Examples

1. `sp_helpcache pub_cache`

   Displays information about items bound to *pub_cache*.

2. `sp_helpcache "80M"`

   Shows the amount of overhead required to create an 80MB data cache.

3. `sp_helpcache`

   Displays information about all caches and all items bound to them.

Comments

- To see the size, status, and I/O size of all data caches on the server, use **sp_cacheconfig**.
- When you configure data caches with **sp_cacheconfig**, all the memory that you specify is made available to the data cache. Overhead for managing the cache is taken from the default data cache. The **sp_helpcache** displays the amount of memory required for a cache of the specified size.
- To bind objects to a cache, use **sp_bindcache**. To unbind a specific object from a cache, use **sp_unbindcache**. To unbind all objects that are bound to a specific cache, use **sp_unbindcache_all**.

- The procedure **sp_cacheconfig** configures data caches. The procedure **sp_poolconfig** configures memory pools within data caches.
- **sp_helpcache** computes overhead accurately up to 74GB.

Permissions

Any user can execute **sp_helpcache.**

Tables Used

master..sysattributes, *master..sysdatabases*, *sysattributes*, *sysindexes*, *sysobjects*

See Also

System procedures	**sp_bindcache**, **sp_cacheconfig**, **sp_poolconfig**, **sp_unbindcache**, **sp_unbindcache_all**

sp_helpconfig

Function

Reports help information on configuration parameters.

Syntax

sp_helpconfig "*configname*", ["*size*"]

Parameters

configname – is the configuration parameter being queried, or a non-unique parameter fragment.

size – is the size of memory, specified by **B** (bytes), **K** (kilobytes), **M** (megabytes), **G** (gigabytes), or **P** (pages). Used without the type of size specified, *size* specifies the number of the entity being configured using this parameter, for examples, locks, open indexes, and so on. *size* is ignored if *configname* is not a unique parameter name.

Examples

1. sp_helpconfig "allow"

```
Configuration option is not unique.
 option_name                    config_value run_value
 ------------------------------ ------------ -----------
 allow backward scans                      1           1
 allow nested triggers                     1           1
 allow procedure grouping                  1           1
 allow remote access                       1           1
 allow resource limits                     0           0
 allow sendmsg                             0           0
 allow sql server async i/o                1           1
 allow updates to system tables            0           0
```

2. sp_helpconfig "open objects", "421"

```
number of open objects sets the maximum number of database objects
that are open at one time on SQL Server. The default run value is 500.

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
          100     2147483647            500            500          243

Configuration parameter, 'number of open objects', will consume 207K
of memory if configured at 421.
```

Returns a report on how much memory is needed to create a metadata cache for 421 object descriptors.

3. sp_helpconfig "open databases", "1M"

```
number of open databases sets the maximum number of databases that can
be open at one time on SQL Server. The default run value is 12.

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
            5     2147483647             12             12          433

Configuration parameter, 'number of open databases', can be configured
to 28 to fit in 1M of memory.
```

Returns a report on how many database descriptors would fill a 1MB database cache.

4. sp_helpconfig "number of locks", "512K"

```
number of locks sets the number of available locks. The default run
value is 5000.

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
         1000    2147483647           5000           5000          528

Configuration parameter 'number of locks', can be configured to 4848
to fit in 512K of memory.
```

Returns a report on how many locks will use 512K of memory.

5. sp_helpconfig "allow updates to system tables"

```
allow updates to system tables allows system tables to be updated
directly. The default is 0 (off).

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
            0              1              0              0            0
```

Returns a report on the status of the **allow updates to system tables** configuration parameter.

Comments

- **sp_helpconfig** reports help information on configuration parameters, such as how much memory would be needed if the parameter were set to a certain value. **sp_helpconfig** also displays the current setting, the amount of memory used for that setting, the default value, and the minimum and maximum settings.

- If you use a nonunique parameter fragment for *configname*, **sp_helpconfig** returns a list of matching parameters with their configured values and current values. See example 1.

Planning metadata cache configuration

- Use **sp_helpconfig** when you are planning a metadata cache configuration for a server.

  For example, suppose you were planning to move a database that contained 2000 user indexes to a different server. To find how much memory you would need to configure for that server so that it would accommodate the database's user indexes, enter the following command:

  sp_helpconfig "open indexes", "2000"

```
number of open indexes sets the maximum number of indexes that can be
open at one time on SQL Server. The default run value is 500.

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
          100     2147483647            500            500          208

Configuration parameter, 'number of open indexes', will consume 829k
of memory if configured at 2000.
```

  Alternatively, suppose you had 1MB of memory available for the index cache, and you needed to know how many index descriptors it would support. Run the following command:

  sp_helpconfig "open indexes", "1M"

```
number of open indexes sets the maximum number of indexes that can be
open at one time on SQL Server. The default run value is 500.

Minimum Value  Maximum Value  Default Value  Current Value  Memory Used
-------------  -------------  -------------  -------------  -----------
          100     2147483647            500            500          208

Configuration parameter 'number of open indexes', can be configured to
2461 to fit in 1M of memory.
```

  Based on this output, if you have 1MB of memory, you can create an index descriptor cache that can contain a maximum of 2461 index descriptors. To create this cache, set the **number of open indexes** configuration parameter as follows:

  sp_configure "number of open indexes", 2461

Using *sp_helpconfig* With *sybdiagdb*

Sybase Technical Support may create the *sybdiagdb* database on your system for debugging purposes. This database holds diagnostic configuration data, and is for use by Sybase Technical Support only.

The following *configname* options have been added to **sp_helpconfig** for use with the *sybdiagdb* database:

- *number of ccbs* is the number of configurable action point control blocks available to aid debugging.
- *caps per ccb* is the maximum number of configurable action points that can be configured at any one time within one configurable action point.
- *average cap size* is the estimated number of bytes of memory required to store the information associated with a typical configurable action point.

For example:

```
sp_helpconfig "number of ccbs"
Minimum Value Maximum Value Default Value Current Value Memory Used
------------- ------------- ------------- ------------- -----------
0             100           0             0             0

sp_helpconfig "caps per ccb"
Minimum Value Maximum Value Default Value Current Value Memory Used
------------- ------------- ------------- ------------- -----------
5             500           50            50            0

sp_helpconfig "average cap size"
Minimum Value Maximum Value Default Value Current Value Memory Used
------------- ------------- ------------- ------------- -----------
100           10000         200           200           0
```

Permissions

The options specified in “Using sp_helpconfig With sybdiagdb” can be used only by Sybase Technical Support. Any user can execute **sp_helpconfig** with other *configname* options.

Tables Used

sysindexes, sysobjects, sysdatabases

Diagnostic Parameters

The following three configuration parameters are part of **sp_helpconfig** and are to be used by Sybase Technical Support, for diagnostic purposes only, with the *sybdiagdb* database:

number of ccbs is the number of configurable action point control blocks available to aid debugging.

caps per ccb is the maximum number of configurable action points that can be configured at any one time within one configurable action point.

average cap size is the estimated number of bytes of memory required to store the information associated with a typical configurable action point.

Syntax

```
sp_helpconfig “number of ccbs”
sp_helpconfig “caps per ccb”
sp_helpconfig “average cap size”
```

Examples

```
> sp_helpconfig “number of ccbs”
2> go
 Minimum Value Maximum Value Default Value Current Value Memory Used
 ------------- ------------- ------------- ------------- -----------
             0           100             0             0           0
 (return status = 0)


1> sp_helpconfig “caps per ccb”
2> go
 Minimum Value Maximum Value Default Value Current Value Memory Used
 ------------- ------------- ------------- ------------- -----------
             5           500            50            50           0
 (return status = 0)

1> sp_helpconfig “average cap size”
2> go
```

```
Minimum Value Maximum Value Default Value Current Value Memory Used
------------- ------------- ------------- ------------- -----------
          100         10000           200           200           0
(return status = 0)
```

```
sp_helpconfig “average cap size”
```

See Also

System procedures	**sp_configure**, **sp_countmetadata**, **sp_monitorconfig**

sp_helpconstraint

Function

Reports information about integrity constraints used in the specified tables.

Syntax

sp_helpconstraint [*objname*] [, detail]

Parameters

objname – is the name of a table that has one or more integrity constraints defined by a **create table** or **alter table** statement.

detail – returns information about the constraint's user or error messages.

Examples

1. sp_helpconstraint store_employees

```
name                          defn
----------------------------  --------------------------------
store_empl_stor_i_272004000   store_employees FOREIGN KEY
                              (stor_id) REFERENCES stores(stor_id)
store_empl_mgr_id_288004057   store_employees FOREIGN KEY
                              (mgr_id) SELF REFERENCES
                              store_employees(emp_id)
store_empl_2560039432         UNIQUE INDEX( emp_id) :
                              NONCLUSTERED, FOREIGN REFERENCE

(3 rows affected)

Total Number of Referential Constraints: 2

Details:
-- Number of references made by this table: 2
-- Number of references to this table: 1
-- Number of self references to this table: 1

Formula for Calculation:
Total Number of Referential Constraints
= Number of references made by this table
+ Number of references made to this table
- Number of self references within this table
```

Displays the constraint information for the *store_employees* table in the *pubs3* database. The *store_employees* table has a foreign key to the *stores* table (*stor_id*) and a self-reference (*mgr_id* references *emp_id*).

2. sp_helpconstraint titles, detail

```
name                           type
      defn
        msg
-----------------------------  ------------------------
     ------------------------------------------------------------
       ------------------------------------------
datedflt                       default value
      create default datedflt as getdate()

typedflt                       default value
      create default typedflt as "UNDECIDED"

titles_pub_id_96003373         referential constraint
      titles FOREIGN KEY (pub_id) REFERENCES publishers(pub_id)
        standard system error message number : 547

roysched_title__144003544      referential constraint
      roysched FOREIGN KEY (title_id) REFERENCES titles(title_id)
        standard system error message number : 547

salesdetai_title__368004342    referential constraint
      salesdetail FOREIGN KEY (title_id) REFERENCES titles(title_id)
        standard system error message number : 547

titleautho_title__432004570    referential constraint
      titleauthor FOREIGN KEY (title_id) REFERENCES titles(title_id)
        standard system error message number : 547

titles_800033162               unique constraint
      UNIQUE INDEX ( title_id) : NONCLUSTERED, FOREIGN REFERENCE
        standard system error message number : 2601

(7 rows affected)

Total Number of Referential Constraints: 4

Details:
-- Number of references made by this table: 1
-- Number of references to this table: 3
-- Number of self references to this table: 0
```

```
Formula for Calculation:
Total Number of Referential Constraints
= Number of references made by this table
+ Number of references made to this table
- Number of self references within this table.
```

Displays more detailed information about the *pubs3..salesdetail* constraints, including the constraint type and any constraint error messages.

3. sp_helpconstraint

```
id          name                      Num_referential_constraints
----------- ------------------------- ---------------------------
   80003316 titles                                              4
   16003088 authors                                             3
  176003658 stores                                              3
  256003943 salesdetail                                         3
  208003772 sales                                               2
  336004228 titleauthor                                         2
  896006223 store_employees                                     2
   48003202 publishers                                          1
  128003487 roysched                                            1
  400004456 discounts                                           1
  448004627 au_pix                                              1
  496004798 blurbs                                              1

(11 rows affected)
```

Displays a listing of all tables in the *pubs3* database.

Comments

- **sp_helpconstraint** prints the name and definition of the integrity constraint, and the number of references used by the table. The **detail** option returns information about the constraint's user or error messages.
- Running **sp_helpconstraint** with no parameters lists all the tables containing references in the current database, and displays the total number of references in each table. **sp_helpconstraint** lists the tables in descending order, based on the number of references in each table.
- **sp_helpconstraint** reports only the integrity constraint information about a table (defined by a **create table** or **alter table** statement). It does not report information about rules, triggers, or indexes created using the **create index** statement. Use **sp_help** to see information about rules, triggers, and indexes for a table.

- For constraints that do not have user-defined messages, Adaptive Server reports the system error message associated with the constraint. Query *sysmessages* to obtain the actual text of that error message.
- You can use **sp_helpconstraint** only for tables in the current database.
- If a query exceeds the configured number of auxiliary scan descriptors, Adaptive Server returns an error message. You can use **sp_helpconstraint** to determine the necessary number of scan descriptors. For more information, see the description of the **number of aux scan descriptors** configuration parameter in the *System Administration Guide.*
- A System Security Officer can prevent the source text of constraint definitions from being displayed to most users who execute **sp_helpconstraint**. To restrict **select** permission on the *text* column of the *syscomments* table to the object owner or a System Administrator, use **sp_configure** to set the **select on syscomments.text column** parameter to **0**. This restriction is required to run Adaptive Server in the evaluated configuration. For more information about the evaluated configuration, see the *System Administration Guide.*

Permissions

Any user can execute **sp_helpconstraint**.

Tables Used

syscolumns, *syscomments*, *sysconstraints*, *sysindexes*, *sysobjects*, *sysreferences*, *sysusermessages*

See Also

Commands	**alter table**, **create table**
System procedures	**sp_help, sp_helpdb, sp_monitorconfig**

sp_helpdb

Function

Reports information about a particular database or about all databases.

Syntax

sp_helpdb [*dbname*]

Parameters

dbname – is the name of the database on which to report information. Without this optional parameter, **sp_helpdb** reports on all databases.

Examples

1. sp_helpdb

```
name           db_size  owner dbid created        status
-------------- -------- ----- ---- -------------- ------------------
master         5.0 MB   sa       1 Jan 01, 1900   no options set
model          2.0 MB   sa       3 Jan 01, 1900   no options set
pubs2          2.0 MB   sa       6 Sep 20, 1995   no options set
sybsystemprocs 16.0 MB  sa       4 Sep 20, 1995   trunc log on chkp
tempdb         2.0 MB   sa       2 Sep 20, 1995   select into/bulkcopy
```

Displays information about all the databases in Adaptive Server.

2. sp_helpdb pubs2

(Not issued from *pubs2*.)

```
name   db_size  owner  dbid  created         status
-----  -------  -----  ----  ------------  ----------------------
pubs2  2.0 MB    sa     4    Mar 05, 1993  abort tran when log full

device_fragments      size    usage           free kbytes
-----------------     ------  ------------  -------------
master                2.0 MB  data and log            576

name    attribute_class attribute     int_value char_value  comments
------- --------------- ------------- --------- ----------- --------
pubs2   buffer manager  cache binding         1 pubs2_cache     NULL
```

Displays information about the *pubs2* database.

3. sp_helpdb pubs2

(Issued from *pubs2*.)

```
name   db_size  owner  dbid  created       status
-----  -------  -----  ----  ------------  ----------------------
pubs2  2.0 MB   sa     4     Mar 05, 1993  abort tran when log full

device_fragments   size    usage         free kbytes
-----------------  ------  ------------  -------------
master             2.0 MB  data and log            576

device                         segment
------------------------------ ------------------------------
master                         default
master                         logsegment
master                         system

name    attribute_class attribute     int_value char_value  comments
------- --------------- ------------- --------- ----------- --------
pubs2   buffer manager  cache binding         1 pubs2_cache     NULL
```

Displays information about the *pubs2* database, and includes segment information.

4. sp_helpdb pubtune

```
name                           attribute_class
attribute                      int_value
         char_value
         comments
-------------------------------------------------------------
pubtune                        lock strategy
       row lock promotion                    NULL
       PCT = 95, LWM = 300, HWM = 300
```

Displays the row lock promotion attributes set for the *pubtune* database.

Comments

- **sp_helpdb** reports on the specified database when *dbname* is given or on all the databases listed in *master.dbo.sysdatabases* when no parameter is supplied.
- Executing **sp_helpdb** *dbname* from *dbname* includes free space and segment information in the report.
- **sp_helpdb** displays information about a database's attributes, giving the attribute's class, name, integer value, character value, and comments, if any attributes are defined. Example 3 shows cache binding attributes for the *pubs2* database.
- **sp_helpdb** reports if a database is offline.
- **sp_helpdb** reports row lock promotion thresholds, if any are defined for the database.

- A database created with the **for load** option has a status of "don't recover" in the output from **sp_helpdb**.
- When Component Integration Services is enabled, **sp_helpdb** lists the default storage location for the specified database or all databases. If there is no default storage location, the display indicates "NULL".

Permissions

Any user can execute **sp_helpdb**.

Tables Used

master.dbo.spt_values, master.dbo.sysattributes, sysdatabases, sysdevices, syslogins, sysmessages, syssegments, sysusages

See Also

Commands	**alter database**, **create database**
System procedures	**sp_configure**, **sp_dboption**, **sp_renamedb**

sp_helpdevice

Function

Reports information about a particular device or about all Adaptive Server database devices and dump devices.

Syntax

sp_helpdevice [*devname*]

Parameters

devname – is the name of the device about which to report information. If you omit this parameter, **sp_helpdevice** reports on all devices.

Examples

1. sp_helpdevice

```
device_name  physical_name   description
-----------  --------------  ----------------------------------------
diskdump     null            disk, dump device
master       d_master        special, default disk, dsync on,physical
                                                          disk, 10 MB

status       cntrltype          device_number          low       high
------       ----------         -------------          ----      -----
16           2                  0                      0         20000
3            0                  0                      0         5120
```

Displays information about all the devices on Adaptive Server.

2. sp_helpdevice diskdump

Reports information about the dump device named *diskdump*.

Comments

- **sp_helpdevice** displays information on the specified device, when *devname* is given, or on all devices in *master.dbo.sysdevices*, when no argument is given.
- The *sysdevices* table contains dump devices and database devices.

  Database devices can be designated as default devices, which means that they can be used for database storage. This can occur when a user issues **create database** or **alter database** and does not specify a database device name or gives the keyword **default**. To make a database device a default database device, execute the system procedure **sp_diskdefault**.

- Add database devices to the system with **disk init**. Add dump devices with **sp_addumpdevice**.
- The number in the "status" column corresponds to the status description in the "description" column.

  The "cntrltype" column specifies the controller number of the device. The "cntrltype" is 2 for disk or file dump devices and 3–8 for tape dump devices. For database devices, the "cntrltype" is usually 0 (unless your installation has a special type of disk controller).

  The "device_number" column is 0 for dump devices, 0 for the master database device, and between 1 and 255 for other database devices. **sp_helpdevice** may report erroneous negative numbers for device numbers greater than 126.

  The "low" and "high" columns represent virtual page numbers, each of which is unique among all the devices in Adaptive Server.

Permissions

Any user can execute **sp_helpdevice**.

Tables Used

master.dbo.spt_values, *sysdevices*, *sysmessages*

See Also

Commands	**disk init**, **dump database**, **dump transaction**, **load database**, **load transaction**
System procedures	**sp_addumpdevice**, **sp_configure**, **sp_deviceattr**, **sp_diskdefault**, **sp_dropdevice**, **sp_helpdb**, **sp_logdevice**, **sp_who**

sp_helpextendedproc

Function

Displays extended stored procedures (ESPs) in the current database, along with their associated DLL files.

Syntax

sp_helpextendedproc [*esp_name*]

Parameters

esp_name – is the name of the extended stored procedure. It must be a procedure in the current database.

Examples

1.
```
use sybsystemprocs
go
sp_helpextendedproc xp_cmdshell
```

```
ESP Name    DLL Name
----------- ----------
xp_cmdshell sybsyesp
```

Lists the **xp_cmdshell** ESP and the name of the DLL file in which its function is stored.

2.
```
sp_helpextendedproc
```

```
ESP Name    DLL Name
----------- ----------
xp_freedl    sybsyesp
xp_cmdshell  sybsyesp
```

Lists all the ESPs in the current database, along with the names of the DLL files in which their functions are stored.

Comments

- If the *esp_name* is omitted, **sp_helpextendedproc** lists all the extended stored procedures in the database.
- The *esp_name* is case sensitive. It must match the *esp_name* used to create the ESP.

Permissions

Only a System Administrator can execute **sp_helpextendedproc** to see all the ESPs in the database. All users can execute **sp_helpextendedproc** to see ESPs owned by themselves or by the Database Owner.

Tables Used

master.dbo.syscomments, *sysobjects*

See Also

Commands	**create procedure, drop procedure**
System procedures	**sp_addextendedproc**, **sp_dropextendedproc**

sp_helpexternlogin

(Component Integration Services only)

Function

Reports information about external login names.

Syntax

```
sp_helpexternlogin [remote_server [, login_name]]
```

Parameters

remote_server – is the name of the remote server that has been added to the local server with **sp_addserver**.

login_name – is a login account on the local server.

Examples

1. `sp_helpexternlogin`

   Displays all remote servers, local login names, and external logins.

2. `sp_helpexternlogin SSB`

   Displays local login names and external logins for the server named SSB.

3. `sp_helpexternlogin NULL, milo`

   Displays remote servers, local login names and external logins for the user named “milo”.

4. `sp_helpexternlogin SSB, trixi`

   Displays external logins for remote server SSB where the local user name is “trixi”.

Comments

- **sp_helpexternlogin** displays all remote servers, the user’s local login name, and the user’s external login name.
- Add remote servers with **sp_addserver**. Add local logins with **sp_addlogin**.

Permissions

Any user can execute **sp_helpexternlogin**.

Tables Used

master.dbo.syslogins, *master.dbo.sysattributes*, *master.dbo.sysservers*

See Also

System procedures	**sp_addexternlogin**, **sp_addlogin**,**sp_addserver**, **sp_helpserver**

sp_helpgroup

Function

Reports information about a particular group or about all groups in the current database.

Syntax

sp_helpgroup [*grpname*]

Parameters

grpname – is the name of a group in the database created with **sp_addgroup**.

Examples

1. sp_helpgroup

```
Group_name      Group_id
--------------- --------
hackers         16384
public          0
```

Displays information about all groups in the current database.

2. sp_helpgroup hackers

```
Group_name  Group_id    Users_in_group  Userid
----------- ---------   --------------  ------
hackers     16384       ann             4
hackers     16384       judy            3
```

Displays information about the group “hackers”.

Comments

- To get a report on the default group, “public,” enclose the name “public” in single or double quotes (“public” is a reserved word).
- If there are no members in the specified group, **sp_helpgroup** displays the header, but lists no users, as follows:

```
Group_name  Group_id    Users_in_group  Userid
----------- ---------   --------------  ------
```

Permissions

Any user can execute **sp_helpgroup**.

Tables Used

syssrvroles, *sysusers*

See Also

Commands	**grant**, **revoke**
System procedures	**sp_addgroup**, **sp_changegroup**, **sp_dropgroup**, **sp_helprotect**, **sp_helpuser**

sp_helpindex

Function

Reports information about the indexes created on a table.

Syntax

sp_helpindex *objname*

Parameters

objname – is the name of a table in the current database.

Examples

1. sp_helpindex sysobjects

```
index_name           index_description
      index_keys
      index_max_rows_per_page index_fillfactor index_reservepagegap
 ------------------- ------------------------------------------------------
        ---------------------- ---------------- --------------------
 sysobjects           clustered, unique located on system
     id
                             0                0                    0
 ncsysobjects         nonclustered, unique located on system
     name,uid
                             0                0                    0
```

Displays the types of indexes on the *sysobjects* table.

2. sp_helpindex titles

```
index_name       index_description
   index_keys
   index_max_rows_per_page index_fillfactor index_reservepagegap
---------------- ---------------------------------------------
title_id_ix      nonclustered, unique located on default
    title_id
                        0                0                    0
publ_ix          nonclustered located on default
    pub_id, pubdate DESC
                        0                0                    8
title_ix         clustered, allow duplicate rows located on default
    title
                        0               90                    0
```

The index on *publ_ix* was created with *pub_id* in ascending order and *pubdate* in descending order.

Comments

- **sp_helpindex** lists any indexes on a table, including indexes created by defining unique or primary key constraints defined by a **create table** or **alter table** statement.
- **sp_helpindex** displays any attributes (for example, cache bindings) assigned to the indexes on a table.
- **sp_helpindex** displays:
  - The **max_rows_per_page** setting of the indexes.
  - Information about clustered indexes on data-only locked tables

    The index ID (*indid)* of a clustered index in data-only locked tables is not equal to 1.
  - The column order of the keys, to indicate whether they are in ascending or descending order.
  - Space manage property values
  - The key column name followed by the order. Only descending order is displayed. For example, if there is an index on column a ASC, b DESC, c ASC, "index_keys" shows "a, b DESC, c".

Permissions

Any user can execute **sp_helpindex**.

Tables Used

master.dbo.spt_values, *sysattributes*, *sysindexes*, *sysobjects*, *syssegments*

See Also

Commands	**create index**, **drop index**, **update statistics**
System procedures	**sp_help**, **sp_helpkey**

sp_helpjava

Function

Displays information about Java classes and associated JARs that are installed in the database.

Refer to *Java in Adaptive Server Enterprise* for more information about Java in the database.

Syntax

```
sp_helpjava ["class" [, java_class_name [, detail]]
  | "jar" [, jar_name] ]
```

Parameters

"class" | "jar" – specifies whether to display information about a class or a JAR. Both "class" and "jar" are keywords, so the quotes are required.

java_class_name – the name of the class about which you want information. The class must be a system class or a user-defined class that is installed in the database.

detail – specifies that you want to see detailed information about the class.

jar_name – the name of the JAR for which you want to see information. The JAR must be installed in the database using **installjava**.

Examples

1. **sp_helpjava**

   Displays the names of all classes and associated JAR files installed in the database.

2. **sp_helpjava "class"**

   Displays the name of all classes.

3. **sp_helpjava "class", Address, detail**

   Displays detailed information about the **Address** class. For example:

```
Class
 -------------------------------------------------
 Address

(1 row affected)
 Class Modifiers
 -------------------------------------------------
 public synchronized

 Implemented Interfaces
 -------------------------------------------------
 java.io.Serializable

 Extended Superclass
 -------------------------------------------------
 java.lang.Object

 Constructors
 -------------------------------------------------
 public Address()
 public Address(java.lang.String,java.lang.String)

 Methods
 -------------------------------------------------
 public final native java.lang.Class
java.lang.Object.getClass()
 public native int java.lang.Object.hashCode()
 public boolean
java.lang.Object.equals(java.lang.Object)
 public java.lang.String
java.lang.Object.toString()
 public final native void java.lang.Object.notify()
 public final native void
java.lang.Object.notifyAll()
 public final native void
java.lang.Object.wait(long) throws
java.lang.InterruptedException
 public final void java.lang.Object.wait(long,int)
throws java.lang.InterruptedException
 public final void java.lang.Object.wait() throws
java.lang.InterruptedException
 public java.lang.String Address.display()
 public void Address.removeLeadingBlanks()

 Fields
 -------------------------------------
 public java.lang.String Address.street
 public java.lang.String Address.zip
```

Permissions

Any user can execute **sp_helpjava.**

Tables Used

sysjars, *sysxtypes*

See Also

Commands	**remove java**
Utilities	**extractjava, installjava**

sp_helpjoins

Function

Lists the columns in two tables or views that are likely join candidates.

Syntax

sp_helpjoins *lefttab*, *righttab*

Parameters

lefttab – is the first table or view.

righttab – is the second table or view. The order of the parameters does not matter.

Examples

1. sp_helpjoins sales, salesdetail

```
a1       a2       b1       b2       c1       c2
   d1       d2       e1       e2       f1       f2
      g1       g2       h1       h2
-------- -------- -------- -------- -------- --------
   -------- -------- -------- -------- -------- --------
      -------- -------- -------- --------
stor_id  stor_id  ord_num  ord_num  NULL     NULL
   NULL     NULL     NULL     NULL     NULL     NULL
      NULL     NULL     NULL     NULL
```

Displays a list of columns that are likely join candidates in the *sales* and *salesdetail* tables.

2. sp_helpjoins sysobjects, syscolumns

```
a1   a2   b1   b2   c1   c2   d1   d2   e1   e2
        f1   f2   g1   g2   h1   h2
---- ---- ---- ---- ---- ---- ---- ---- ---- ----
        ---- ---- ---- ---- ---- ----
id   id   NULL NULL NULL NULL NULL NULL NULL NULL
        NULL NULL NULL NULL NULL NULL
```

Displays a list of columns that are likely join candidates in the *sysobjects* and *syscolumns* system tables.

Comments

- The column pairs that **sp_helpjoins** displays come from either of two sources. **sp_helpjoins** checks the *syskeys* table in the current database to see if any foreign keys have been defined with

sp_foreignkey on the two tables, then checks to see if any common keys have been defined with **sp_commonkey** on the two tables. If **sp_helpjoins** does not find any foreign keys or common keys there, it checks for keys with the same user-defined datatypes. If that fails, it checks for columns with the same name and datatype.

- **sp_helpjoins** does not create any joins.

Permissions

Any user can execute **sp_helpjoins**.

Tables Used

syscolumns, syskeys, sysobjects

See Also

System procedures	**sp_commonkey**, **sp_foreignkey**, **sp_help**, **sp_helpkey**, **sp_primarykey**

sp_helpkey

Function

Reports information about a primary, foreign, or common key of a particular table or view, or about all keys in the current database.

Syntax

sp_helpkey [*tabname*]

Parameters

tabname – is the name of a table or view in the current database. If you do not specify a name, the procedure reports on all keys defined in the current database.

Examples

1. sp_helpkey

```
keytype object      related_object object_keys         related_keys
------- -------     -------------- ---------------     --------------
primary authors     -- none --     au_id,*,*,*,*,*,*,*  *,*,*,*,*,*,*,*
foreign titleauthor authors        au_id,*,*,*,*,*,*,* au_id,*,*,*,*,*,
                                                          *,*
```

Displays information about the keys defined in the current database. The "object_keys" and "related_keys" columns refer to the names of the columns that make up the key.

Comments

- **sp_helpkey** lists information about all primary, foreign, and common key definitions that reference the table *tabname* or, if *tabname* is omitted, about all the keys in the database. Define these keys with the **sp_primarykey**, **sp_foreignkey**, and **sp_commonkey** system procedures.
- **sp_helpkey** does not provide information about the **unique** or **primary key** integrity constraints defined by a **create table** statement. Use **sp_helpconstraint** to determine what constraints are defined for a table.
- Create keys to make explicit a logical relationship that is implicit in your database design so that applications can use the information.

- If you specify an object name, **sp_helpkey** follows the Adaptive Server rules for finding objects:
  - If you do not specify an owner name, and you own an object with the specified name, **sp_helpkey** reports on that object.
  - If you do not specify an owner name, and you do not own an object of that name, but the Database Owner does, **sp_helpkey** reports on the Database Owner's object.
  - If neither you nor the Database Owner owns an object with the specified name, **sp_helpkey** reports an error condition, even if an object with that name exists in the database for a different owner.
  - If both you and the Database Owner own objects with the specified name, and you want to access the Database Owner's object, specify the name in the form *dbo.objectname*.
- Qualify objects that are owned by database users other than yourself and the Database Owner with the owner's name, as in "mary.myproc".

Permissions

Any user can execute **sp_helpkey.**

Tables Used

master.dbo.spt_values, *syskeys*, *sysobjects*

See Also

Commands	**create trigger**
System procedures	**sp_commonkey**, **sp_foreignkey**, **sp_help**, **sp_primarykey**

sp_helplanguage

Function

Reports information about a particular alternate language or about all languages.

Syntax

sp_helplanguage [*language*]

Parameters

language – is the name of the alternate language you want information about.

Examples

1. sp_helplanguage french

```
langid dateformat datefirst upgrade     name
       alias
       months
       shortmonths
       days
------ ---------- --------- ----------- ----------------------
      ----------------------------
      ------------------------------------------------------------
      ------------------------------------------------------------
      ------------------------------------------------------------
1     dmy        1         0         french
      french
      janvier,février,mars,avril,mai,juin,juillet,août,septembre,
          octobre,novembre,décembre
      jan,fév,mar,avr,mai,jui,juil,aoû,sep,oct,nov,déc
      lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche
```

Displays information about the alternate language, “french”.

2. sp_helplanguage

Displays information about all installed alternate languages.

Comments

- **sp_helplanguage** reports on a specified language, when the language is given, or on all languages in *master.dbo.syslanguages*, when no language is supplied.

Permissions

Any user can execute **sp_helplanguage**.

Tables Used

master.dbo.syslanguages

See Also

System procedures	**sp_addlanguage**, **sp_droplanguage**, **sp_setlangalias**

sp_helplog

Function

Reports the name of the device that contains the first page of the transaction log.

Syntax

```
sp_helplog
```

Parameters

None.

Examples

```
1. sp_helplog
```

```
In database 'master', the log starts on device
'master'.
```

Comments

- **sp_helplog** displays the name of the device that contains the first page of the transaction log in the current database.

Permissions

Any user can execute **sp_helplog**.

Tables Used

master.dbo.sysdevices, *master.dbo.sysusages*, *sysindexes*, *sysobjects*

See Also

Commands	**alter database**, **create database**
System procedures	**sp_helpdevice**, **sp_logdevice**

sp_helpobjectdef

(Component Integration Services only)

Function

Reports owners, objects, and type information for remote object definitions.

Syntax

```
sp_helpobjectdef [object_name]
```

Parameters

object_name – is the name of the object as it is defined in the *sysattributes* table. The *object_name* can be in any of the following forms:

- *dbname.owner.object*
- *dbname..object*
- *owner.object*
- *object*

dbname and *owner* are optional. *object* is required. If *owner* is not supplied, the *owner* defaults to the current user name. If *dbname* is supplied, it must be the current database, and *owner* must be supplied or marked with the placeholder *dbname..object*. Enclose a multipart *object_name* in quotes.

Examples

1. `sp_helpobjectdef`

   Displays all remote object definitions in the current database.

2. `sp_helpobjectdef "dbo.tb1"`

   Displays remote object definitions for the *tb1* table owned by the Database Owner.

Comments

- If no *object_name* is supplied, **sp_helpobjectdef** displays all remote object definitions.
- A server name is not permitted in the *object_name* parameter.

Permissions

Any user can execute **sp_helpobjectdef.**

Tables Used

sysattributes, sysobjects, sysservers, spt_values

See Also

Commands	**create table, create existing table, drop table**
System procedures	**sp_addlogin**, **sp_addserver**, **sp_addobjectdef**, **sp_defaultloc**, **sp_dropobjectdef**, **sp_help**, **sp_helpserver**

sp_help_qpgroup

Function

Reports information on an abstract plan group.

Syntax

```
sp_help_qpgroup [ group [, mode ]]
```

Parameters

group – is the name of an abstract plan group.

mode – is the type of report to print, one of the following:

Mode	Information Returned
full	The number of rows and number of plans in the group, the number of plans that use two or more rows, the number of rows and plan IDs for the longest plans, and number of hash keys and hash key collision information. This is the default report mode.
stats	All of the information from the "full" report, except hash key information.
hash	The number of rows and number of abstract plans in the group, the number of hash keys, and hash-key collision information.
list	The number of rows and number of abstract plans in the group, and the following information for each query / plan pair: hash key, plan ID, first few characters of the query, and the first few characters of the plan.
queries	The number of rows and number of abstract plans in the group, and the following information for each query: hash key, plan ID, first few characters of the query.
plans	The number of rows and number of abstract plans in the group, and the following information for each plan: hash key, plan ID, first few characters of the plan.
counts	The number of rows and number of abstract plans in the group, and the following information for each plan: number of rows, number of characters, hash key, plan ID, first few characters of the query.

Examples

1. sp_help_qpgroup

```
Group                   GID         Plans
----------------------- ----------- -----------
ap_stdin                          1           0
ap_stdout                         2           0
dev_test                          3         209
```

Reports summary information about all abstract plan groups in the database.

2. sp_help_qpgroup test_plans

```
Query plans group 'test_plans', GID 8

 Total Rows  Total QueryPlans
 ----------- ----------------
           6                3
sysqueryplans rows consumption, number of query
plans per row count

 Rows        Plans
 ----------- -----------
           2           3

 Hashkeys
 -----------
           3
There is no hash key collision in this group.
```

Reports on the *test_plans* group.

Comments

- When used with an abstract plan group name, and no *mode* parameter, the default mode for **sp_help_qpgroup** is **full**.
- Hash-key collisions indicate that more than one plan for a particular user has the same hash-key value. When there are hash key collisions, the query text of each query with the matching hash key must be compared to the user's query text in order to identify the matching query, so performance is slightly degraded.

Permissions

Any user can execute **sp_help_qpgroup**.

Tables Used

sysattributes, sysqueryplans

See Also

System procedures	**sp_help_qplan**

sp_help_qplan

Function

Reports information about an abstract plan.

Syntax

sp_help_qplan *id* [, *mode*]

Parameters

id – is the ID of the abstract plan.

mode – is the type of report to print, one of the following:

mode	Information returned
full	The plan ID, group ID, and hash key, and the full query and plan text.
brief	The same as full, but only prints about 80 characters of the query and plan, rather than the full query and plan. This is the default mode.
list	The hash key, ID, and first 20 characters of the query and plan.

Examples

1. sp_help_qplan 800005881

```
gid         hashkey     id
 ----------- ----------- -----------
           5  2054169974   937054374

 query
 ----------------------------------------------------
----------------------------
 select type, avg(price) from titles group by type

 plan
 ----------------------------------------------------
----------------------------
 ( plan
    ( store Worktab1
        ( i_scan type_price titles )
    )
    ( t_scan ( ...
```

Prints the brief abstract plan report.

2. `sp_help_qplan 784005824, full`

   Prints the full abstract plan report.

Comments

- If you do not supply a value for the *mode* parameter, the default is **brief**.

Permissions

Any user can execute **sp_help_qplan** to see the abstract plan of a query that he or she owns. Only the System Administrator and the Database Owner can display an abstract plan owned by another user.

Tables Used

sysqueryplans

See Also

System procedures	**sp_find_qplan**, **sp_help_qpgroup**

sp_helpremotelogin

Function

Reports information about a particular remote server's logins or about all remote server logins.

Syntax

```
sp_helpremotelogin [remoteserver [, remotename]]
```

Parameters

remoteserver – is the name of the server about which to report remote login information.

remotename – is the name of a particular remote user on the remote server.

Examples

1. `sp_helpremotelogin GATEWAY`

   Displays information about all the remote users of the remote server GATEWAY.

2. `sp_helpremotelogin`

   Displays information about all the remote users of all the remote servers known to the local server.

Comments

- **sp_helpremotelogin** reports on the remote logins for the specified server, when *remoteserver* is given, or on all servers, when no parameter is supplied.

Permissions

Any user can execute **sp_helpremotelogin.**

Tables Used

master.dbo.spt_values, *master.dbo.sysmessages*, *master.dbo.sysremotelogins*, *master.dbo.sysservers*, *sysobjects*

See Also

System procedures	**sp_addremotelogin**, **sp_dropremotelogin**, **sp_helpserver**

sp_help_resource_limit

Function

Reports on resource limits.

Syntax

```
sp_help_resource_limit [name [, appname [, limittime
   [, limitday [, scope [, action]]]]]]
```

Parameters

name – is the Adaptive Server login to which the limits apply. For information about limits that govern a particular login, specify the login *name*. For information about limits without regard to login, specify **null**.

➤ ***Note***

If you are not a System Administrator, specify your own login, or a login of NULL, to display information about the resource limits that apply to you.

appname – is the name of the application to which the limit applies. For information about limits that govern a particular application, specify the application name that the client program passes to the Adaptive Server in the login packet. For information about limits without regard to application, specify **null**.

limittime – is the time during which the limit is enforced. For information about limits in effect at a given time, specify the time, with a value between "00:00" and "23:59", using the following form:

"*HH*:*MM*"

For information about limits without regard to time, specify **null**.

limitday – is any day on which the limit is enforced. For information about resource limits in effect on a given day of the week, specify the full weekday name for the default server language, as stored in the *syslanguages* system table of the *master* database. For information about limits without regard to the days on which they are enforced, specify **null**.

scope – is the scope of the limit. Specify one of the following:

Scope Code	For Help on All Limits That Govern
1	Queries
2	Query batches (one or more SQL statements sent by the client to the server)
4	Transactions
6	Both query batches and transactions
NULL	The specified *name*, *appname*, *limittime*, *limitday*, and *action*, without regard to their *scope*

action – is the action to take when the limit is exceeded. Specify one of the following:

Action Code	For Help on All Limits That
1	Issue a warning
2	Abort the query batch
3	Abort the transaction
4	Kill the session
NULL	Govern the specified *name*, *appname*, *limittime*, *limitday*, and *scope*, without regard to the *action* they take

Examples

1. **`sp_help_resource_limit`**

   Lists all resource limits stored in the *sysresourcelimits* system table.

2. **`sp_help_resource_limit joe_user`**

   Lists all limits for the user “joe_user”.

3. **`sp_help_resource_limit NULL, my_app`**

   Lists all limits for the application *my_app*.

4. **`sp_help_resource_limit NULL, NULL, "09:00"`**

   Lists all limits enforced at 9:00 a.m.

5. **`sp_help_resource_limit @limittype = "09:00"`**

   An alternative way of listing the limits enforced at 9:00 a.m.

6. **`sp_help_resource_limit NULL, NULL, NULL, Monday`**

   Lists all limits enforced on Mondays.

7. **`sp_help_resource_limit joe_user, NULL, "09:00", Monday`**

   Lists any limit in effect for "joe_user" on Mondays at 9:00 a.m.

Comments

- **sp_help_resource_limit** reports on all resource limits, limits for a given login or application, limits in effect at a given time or day of the week, or limits with a given scope or action.
- For more information on resource limits, see the *System Administration Guide.*

Permissions

Any user can execute **sp_help_resource_limit** to list his or her own resource limits. Only a System Administrator can execute **sp_help_resource_limit** to list limits that apply to other users.

Tables Used

master..sysresourcelimits, master..systimeranges, master..spt_limit_types

See Also

System procedures	**sp_add_resource_limit**, **sp_drop_resource_limit**, **sp_modify_resource_limit**

sp_helprotect

Function

Reports on permissions for database objects, users, groups, or roles.

Syntax

```
sp_helprotect [name [, username [, "grant"
   [,"none"|"granted"|"enabled"|role_name]]]]
```

Parameters

name – is either the name of the table, view, stored procedure, or the name of a user, user-defined role, or group in the current database. If you do not provide a name, **sp_helprotect** reports on all permissions in the database.

username – is a user's name in the current database.

grant – displays the privileges granted to *name* **with grant option.**

none – ignores roles granted to the user when determining permissions granted.

granted – includes information on all roles granted to the user when determining permissions granted.

enabled – includes information on all roles activated by the user when determining permissions granted.

role_name – displays permission information for the specified role only, regardless of whether this role has been granted to the user.

Examples

1. **grant select on titles to judy**
   grant update on titles to judy
   revoke update on titles(price) from judy
   grant select on publishers to judy
   with grant option

   After this series of **grant** and **revoke** statements, executing **sp_helprotect titles** results in this display:

```
grantor grantee type    action   object      column       grantable
------- ------  -----   ------   ------      ------       ---------
dbo     judy    Grant   Select   titles      All          FALSE
dbo     judy    Grant   Update   titles      advance      FALSE
dbo     judy    Grant   Update   titles      notes        FALSE
dbo     judy    Grant   Update   titles      pub_id       FALSE
dbo     judy    Grant   Update   titles      pubdate      FALSE
dbo     judy    Grant   Update   titles      title        FALSE
dbo     judy    Grant   Update   titles      title_id     FALSE
dbo     judy    Grant   Update   titles      total_sales  FALSE
dbo     judy    Grant   Update   titles      type         FALSE
dbo     judy    Grant   Select   publishers  all          TRUE
```

2. **grant select, update on titles(price, advance)**
   to mary
   with grant option
   go
   sp_helprotect titles

   After issuing this **grant** statement, **sp_helprotect** displays the following:

```
grantor   grantee    type    action   object   column   grantable
-------   -------    ------  -------  ------   ------   ---------
dbo       mary       Grant   Select   titles  advance  TRUE
dbo       mary       Grant   Select   titles  price    TRUE
dbo       mary       Grant   Update   titles  advance  TRUE
dbo       mary       Grant   Update   titles  price    TRUE
```

3. **sp_helprotect judy**

   Displays all the permissions that "judy" has in the database.

4. **sp_helprotect sysusers, csmith, null, doctor, "grant"**

   Displays any permissions that "csmith" has on the *sysusers* table, as well as whether "csmith" has **with grant option** which allows "csmith" to grant permissions to other users.

```
grantor   grantee   type   action     object   column   grantable
------------------------------------------------------------------
dbo       doctor    Grant  Delete     sysusers  All     FALSE
dbo       doctor    Grant  Insert     sysusers  All     FALSE
dbo       doctor    Grant  References sysusers  All     FALSE
dbo       doctor    Grant  Select     sysattributes
                                          All     FALSE

(1 row affected)
(return status = 0)
```

5. sp_helpprotect doctor_role

Displays information about the permissions that the doctor role has in the database.

```
grantor   grantee   type   action     object     column grantable
------------------------------------------------------------------
dbo       doctor     Grant  Delete     sysusers    All    FALSE
dbo       doctor     Grant  Insert     sysusers    All    FALSE
dbo       doctor     Grant  References sysusers    All    FALSE
dbo       doctor     Grant  Select     sysattributes
                                          All  FALSE

(1 row affected)
(return status = 0)
```

6. sp_helpprotect sysusers, csmith, null, doctor_role, "granted"

Displays information on all roles granted to “csmith”.

```
grantor grantee   type   action     object    column   grantable
------------------------------------------------------------------
dbo     csmith    Grant   Update     sysusers   All     FALSE
dbo     doctor    Grant   Delete     sysusers   All     FALSE
dbo     doctor    Grant   Insert     sysusers   All     FALSE
dbo     doctor     Grant   References sysusers   All      FALSE

(1 row affected)
(return status = 0)
```

7. sp_helpprotect sysattributes, rpillai, null, intern, "enabled"

Displays information on all active roles granted to “rpillai”.

```
grantor   grantee   type   action   object      column  grantable
--------------- ------------------------------------------------
dbo       public     Grant  Select   sysattributes All  FALSE

(1 row affected)
(return status = 0)
```

Comments

- **sp_helprotect** reports permissions on a database object. If you supply the *username* parameter, only that user's permissions on the database object are reported. If *name* is not an object, **sp_helprotect** checks to see if it is a user, a group, or a role. If it is, **sp_helprotect** lists the permissions for the user, group, or role.
- **sp_helprotect** looks for objects and users in the current database only.
- If you do not specify an optional value such as **granted**, **enabled**, **none**, or *role_name*, Adaptive Server returns information on all roles activated by the current specified user.
- If the specified user is not the current user, Adaptive Server returns information on all roles granted to the specified user.
- Displayed information always includes permissions granted to the group in which the specified user is a member.
- In granting permissions, a System Administrator is treated as the object owner. If a System Administrator grants permission on another user's object, the owner's name appears as the grantor in **sp_helprotect** output.

Permissions

Any user can execute **sp_helprotect** to view his or her own permissions. Only a System Security Officer can execute **sp_helprotect** to view permissions granted to other users.

Tables Used

master.dbo.spt_values, *syscolumns*, *sysobjects*, *sysprotects*, *sysusers*

See Also

Commands	**grant**, **revoke**, **create**, **drop**, **set**
System procedures	**sp_help**, **sp_activeroles**, **sp_configure**, **sp_displaylogin**, **sp_displayroles**, **sp_modifylogin**

sp_helpsegment

Function

Reports information about a particular segment or about all segments in the current database.

Syntax

sp_helpsegment [*segname*]

Parameters

segname – is the name of the segment about which you want information. If you omit this parameter, information about all segments in the current database appears.

Examples

1. sp_helpsegment

```
segment name                          status
------- ------------------------------ ------
      0 system                              0
      1 default                             1
      2 logsegment                          0
```

Reports information about all segments in the current database.

2. sp_helpsegment order_seg

```
segment name                          status
------- ------------------------------ ------
      3 order_seg                         0

device                 size           free_pages
---------------------- -------------- -----------
tpcd_data1             25.0MB               8176
tpcd_data2             25.0MB               8512
tpcd_data3             25.0MB               8392
tpcd_data4             25.0MB               8272
```

```
tpcd_data5              25.0MB                8448
tpcd_data6              25.0MB                8512

table_name              index_name             indid
----------------------- ---------------------- ------
orders                  orders                      0

total_size     total_pages free_pages  used_pages
-------------- ----------- ----------- -----------
150.0MB              76800       50312       26488
```

Reports information about the segment named *order_seg*, including which database tables and indexes use that segment and the total number of pages, free pages and used pages on the segment.

3. sp_helpsegment "default"

Reports information about the *default* segment. The keyword **default** must be enclosed in quotes.

4. sp_helpsegment logsegment

```
segment name                          status
------- ------------------------------ ------
      2 logsegment                          0

device                  size              free_pages
----------------------- ----------------- -----------
tpcd_log1               20.0MB                  10200

table_name              index_name               indid
----------------------- ------------------------ ------
syslogs                 syslogs                       0

total_size        total_pages free_pages  used_pages
----------------- ----------- ----------- -----------
20.0MB                  10240       10200          40
```

Reports information about the segment on which the transaction log is stored.

Comments

- **sp_helpsegment** displays information about the specified segment, when *segname* is given, or about all segments in the current database, when no argument is given.
- When you first create a database, Adaptive Server automatically creates the *system*, *default*, and *logsegment* segments. Use **sp_addsegment** to add segments to the current database.

- The *system*, *default*, and *logsegment* segments are numbered 0, 1, and 2, respectively.
- The "status" column indicates which segment is the default pool of space. Use **sp_placeobject** or the **on** *segment_name* clause of the **create table** or **create index** command to place objects on specific segments.
- The "indid" column is 0 if the table does not have a clustered index and is 1 if the table has a clustered index.

Permissions

Any user can execute **sp_helpsegment**.

Tables Used

master.dbo.sysdevices, *master.dbo.sysusages*, *sysindexes*, *sysobjects*, *syssegments*

See Also

Commands	**create index**, **create table**
System procedures	**sp_addsegment**, **sp_dropsegment**, **sp_extendsegment**, **sp_helpdb**, **sp_helpdevice**, **sp_placeobject**

sp_helpserver

Function

Reports information about a particular remote server or about all remote servers.

Syntax

sp_helpserver [*server*]

Parameters

server – is the name of the remote server about which you want information.

Examples

1. sp_helpserver GATEWAY

Displays information about the remote server GATEWAY.

2. sp_helpserver SYB_BACKUP

```
name       network_name  status                                id
---------- ------------- ----------------------------------- ---
SYB_BACKUP SYB_BACKUP    timeouts, no net password encryption  1
```

Displays information about the local Backup Server.

3. sp_helpserver

Displays information about all the remote servers known to the local server.

Comments

- **sp_helpserver** reports information about all servers in *master.dbo.sysservers* or about a particular remote server, when *server* is specified.
- When Component Integration Services is installed, **sp_helpserver** lists the server class for each server.

Permissions

Any user can execute **sp_helpserver.**

Tables Used

master.dbo.spt_values, *master.dbo.sysservers*, *sysobjects*

See Also

System procedures	**sp_addserver**, **sp_dropserver**, **sp_helpremotelogin**, **sp_serveroption**

sp_helpsort

Function

Displays Adaptive Server's default sort order and character set.

Syntax

sp_helpsort

Parameters

None.

Examples

1. sp_helpsort

For Class 1 (single-byte) character sets, **sp_helpsort** displays the name of the server's default sort order, its character set, and a table of its primary sort values. On a 7-bit terminal, it appears as follows:

```
Sort Order Description
------------------------------------------------------------------
 Character Set = 1, iso_1
     ISO 8859-1 (Latin-1) - Western European 8-bit character set.
 Sort Order = 50, bin_iso_1
     Binary sort order for the ISO 8859/1 character set (iso_1).
Characters, in Order

------------------------------------------------------------------
   ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
   @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
   ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
   ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
   @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
   ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

On an 8-bit terminal, it appears as follows:

```
Sort Order Description
------------------------------------------------------------------
 Character Set = 1, iso_1
     ISO 8859-1 (Latin-1) - Western European 8-bit character set.
 Sort Order = 50, bin_iso_1
     Binary sort order for the ISO 8859/1 character set (iso_1).
Characters, in Order

------------------------------------------------------------------
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
  @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
  ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
  ¡ ¢ £ ¤ ¥ | § ¨ © ª « ¬ –
 ® ¯ ° 2 3 ´ µ ¶ · ¸ 1 º » 1/4 1/2 3/4 ¿ À
  Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï D Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Y P ß à
  á â ã ä å æ ç è é ê ë ì í î ï  ñ ò ó ô õ ö ÷ ø ù ú û ü y p ÿ
```

For a Class 2 (multibyte) character set, the characters are not listed, but a description of the character set is included. For example:

```
Sort Order Description
------------------------------------------------------------------
Character Set = 140, euc_jis
    Japanese. Extended Unix Code mapping for JIS-X0201
    (hankaku katakana) and JIS-X0208 (double byte) roman,
     kana, and kanji.
    Class 2 character set
Sort Order = 50, bin_eucjis
    Binary sort order for Japanese using the EUC JIS
      character set as a basis.
```

Comments

- Binary sort order is the default.

Permissions

Any user can execute **sp_helpsort.**

Tables Used

master.dbo.syscharsets, master.dbo.syscurconfigs, sysobjects

sp_helptext

Function

Displays the **source text** of a **compiled object**.

Syntax

sp_helptext *objname*

Parameters

objname – is the name of the compiled object for which the source text is to be displayed. The compiled object must be in the current database.

Examples

1. sp_helptext pub_idrule

```
# Lines of Text
---------------
1

text
-----------------------------------
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877",
      "1622", "1756")
    or @pub_id like "99[0-9][0-9]"
```

Displays the source text of *pub_idrule*. Since this rule is in the *pubs2* database, execute this command from *pubs2*.

2. sp_helptext sp_helptext

Displays the source text of **sp_helptext**. Since system procedures are stored in *sybsystemprocs*, execute this command from *sybsystemprocs*.

Comments

- **sp_helptext** prints out the number of rows in *syscomments* (255 characters long each) that are occupied by the compiled object, followed by the source text of the compiled object.
- **sp_helptext** looks for the source text in the *syscomments* table in the current database.
- Encrypt the source text with **sp_hidetext**.

- A System Security Officer can prevent the source text of compiled objects from being displayed to most users who execute **sp_helptext**. To restrict **select** permission on the *text* column of the *syscomments* table to the object owner or a System Administrator, use **sp_configure** to set the **select on syscomments.text column** parameter to **0**. This restriction is required to run Adaptive Server in the evaluated configuration. For more information about the evaluated configuration, see the *System Administration Guide*.

Permissions

Any user can execute **sp_helptext**.

Tables Used

syscomments, *sysobjects*

See Also

Commands	**create default**, **create procedure**, **create rule**, **create trigger**, **create view**
System procedures	**sp_checksource**, **sp_help**, **sp_hidetext**

sp_helpthreshold

Function

Reports the segment, free-space value, status, and stored procedure associated with all thresholds in the current database or all thresholds for a particular segment.

Syntax

```
sp_helpthreshold [segname]
```

Parameters

segname – is the name of a segment in the current database.

Examples

1. `sp_helpthreshold logsegment`

   Shows all thresholds on the log segment.

2. `sp_helpthreshold`

   Shows all thresholds on all segments in the current database.

3. `sp_helpthreshold "default"`

   Shows all thresholds on the default segment. Note the use of quotes around the reserved word “default”.

Comments

- **sp_helpthreshold** displays threshold information for all segments in the current database. If you provide the name of a segment, **sp_helpthreshold** lists all thresholds in that segment.
- The *status* column is 1 for the last-chance threshold and 0 for all other thresholds. Databases that do not store their transaction logs on a separate segment have no last-chance threshold.

Permissions

Any user can execute **sp_helpthreshold**.

Tables Used

sysobjects, syssegments, systhresholds

See Also

System procedures	**sp_addthreshold**, **sp_dropthreshold**, **sp_helpsegment**, **sp_modifythreshold**, **sp_thresholdaction**

sp_helpuser

Function

Reports information about a particular user, group, or alias, or about all users, in the current database.

Syntax

sp_helpuser [*name_in_db*]

Parameters

name_in_db – is the user's name in the current database.

Examples

1. sp_helpuser

```
Users_name ID_in_db    Group_name  Login_name
---------  --------    ----------  ----------
ann        4           hackers     ann
dbo        1           public      sa
guest      2           public      NULL
judy       3           hackers     judy
```

Displays information about all users in the current database.

2. sp_helpuser dbo

```
Users_name     ID_in_db    Group_name  Login_name
----------     --------    ----------  ----------
dbo            1           public      sa

Users aliased to user.
Login_name
------------------------------
andy
christa
howard
linda
```

Displays information about the Database Owner (user name "dbo").

Comments

- **sp_helpuser** reports information about all users of the current database. If you specify a *name_in_db*, **sp_helpuser** reports information on the specified user only.

- If the specified user is not listed in the current database's *sysusers* table, **sp_helpuser** checks to see if the user is aliased to another user or is a group name.

Permissions

Any user can execute **sp_helpuser**.

Tables Used

master.dbo.syslogins, *sysalternates*, *sysobjects*, *sysusers*

See Also

Commands	**grant**, **revoke**, **use**
System procedures	**sp_adduser**, **sp_dropuser**, **sp_help**, **sp_helpgroup**

sp_hidetext

Function

Hides the **source text** for the specified **compiled object**.

Syntax

```
sp_hidetext [objname [, tabname [, username]]]
```

Parameters

objname – specifies the compiled object for which to hide the source text.

tabname – specifies the name of the table or view for which to hide the source text.

username – specifies the name of the user who owns the compiled object for which to hide the source text.

Examples

1. **`sp_hidetext`**

   Hides the source text of all compiled objects in the current database.

2. **`sp_hidetext @objname = "sp_sort_table", @username = "Mary"`**

   Hides the source text of the user-defined stored procedure, **sp_sort_table**, that is owned by Mary.

3. **`sp_hidetext "pr_phone_list"`**

   Hides the source text of the stored procedure **pr_phone_list**.

4. **`sp_hidetext @tabname = "my_tab"`**

   Hides the source text of all check constraints, defaults, and triggers defined on the table *my_tab*.

5. **`sp_hidetext "my_vu", "my_tab"`**

   Hides the source text of the view *my_vu* and all check constraints, defaults, and triggers defined on the table *my_tab*.

6. **`sp_hidetext @username = "Tom"`**

   Hides the source text of all compiled objects that are owned by Tom.

Comments

- **sp_hidetext** hides the **source text** for the specified **compiled object**.

◆ ***WARNING!***

Before executing sp_hidetext, make sure you have a backup of the source text. The results of executing sp_hidetext are not reversible.

- If you do not provide any parameters, **sp_hidetext** hides the source text for all compiled objects in the current database.
- For more information about hiding source text, see the *Transact-SQL User's Guide*.

Permissions

Any user can use **sp_hidetext** to hide the source text of his or her own compiled objects. Only a Database Owner or a System Administrator can hide the source text of compiled objects that are owned by another user or use **sp_hidetext** with no parameters.

Tables Used

syscolumns, syscomments, sysconstraints, sysobjects, sysprocedures

See Also

System procedures	**sp_checksource**

sp_import_qpgroup

Function

Imports abstract plans from a user table into an abstract plan group.

Syntax

```
sp_import_qpgroup tab, usr, group
```

Parameters

tab – is the name of a table from which to copy the plans. You can specify a database name, but not an owner name, in the form *dbname..tablename*. The total length must be 30 characters or less.

usr – is the name of the user whose ID should be assigned to the abstract plans when they are imported.

group – is the name of the abstract plan group that contains the plans to be imported.

Examples

1. `sp_import_qpgroup moveplans, dbo, new_plans`

   Copies plans from the table *moveplans* to the *new_plans* group, giving them the user ID for the Database Owner.

Comments

- **sp_import_qpgroup** copies plans from a user table to an abstract plan group in *sysqueryplans*. With **sp_export_qpgroup**, it can be used to copy abstract plan groups between servers and databases, or to copy plans belonging to one user and assign them the ID of another user.
- **sp_import_qpgroup** creates the abstract plan group if it does not exist when the procedure is executed.
- If an abstract plan group exists when **sp_import_qpgroup** is executed, it cannot contain any plans for the specified user. **sp_import_qpgroup** does not check the query text to determine whether queries already exist in the group. If you need to import plans for a user into a group where some plans for the user already exist:
  - Use **sp_import_qpgroup** to import the plans into a new plan group.

  - Use **sp_copy_all_qplans** to copy the plans from the newly-created group to the destination group. **sp_copy_all_qplans** does check queries to be sure that no duplicate plans are created.
  - If you no longer need the group you created for the import, drop the plans in the group with **sp_drop_all_qplans**, then drop the group with **sp_drop_qpgroup**.

- To create an empty table in order to bulk copy abstract plans, use:

```
select * into load_table
from sysqueryplans
where 1 = 2
```

Permissions

Only a System Administrator or the Database Owner can execute **sp_import_qpgroup**.

Tables Used

sysattributes, *sysqueryplans*

See Also

Commands	**create plan**
System procedures	**sp_copy_all_qplans**, **sp_copy_qplan**, **sp_export_qpgroup**,**sp_help_qpgroup**

sp_indsuspect

Function

Checks user tables for indexes marked as suspect during recovery following a sort order change.

Syntax

```
sp_indsuspect [tab_name]
```

Parameters

tab_name – is the name of the user table to be checked.

Examples

```
1. sp_indsuspect newaccts
```

Checks the table *newaccts* for indexes marked as suspect.

Comments

- **sp_indsuspect** with no parameter creates a list of all tables in the current database that have indexes that need to be rebuilt as a result of a sort order change. With a *tab_name* parameter, **sp_indsuspect** checks the specified table for indexes marked as suspect during recovery following a sort order change.
- Use **sp_indsuspect** to list all suspect indexes. The table owner or a System Administrator can use **dbcc reindex** to check the integrity of the listed indexes and to rebuild them if necessary.

Permissions

Any user can execute **sp_indsuspect**.

Tables Used

sysindexes, *sysobjects*, *sysusers*

See Also

Commands	dbcc

sp_listsuspect_db

Function

Lists all databases that currently have offline pages because of corruption detected on recovery.

Syntax

```
sp_listsuspect_db
```

Parameters

None.

Examples

1. sp_listsuspect_db

Lists the databases that have suspect pages.

Comments

- **sp_listsuspect_db** lists the database name, number of suspect pages, and number of objects containing suspect pages.
- Use **sp_listsuspect_page** to identify the suspect pages.

Permissions

Any user can execute **sp_listsuspect_db**.

Tables Used

master.dbo.sysattributes

See Also

System procedures	**sp_forceonline_db, sp_forceonline_page**, **sp_listsuspect_page**, **sp_setsuspect_granularity**, **sp_setsuspect_threshold**

sp_listsuspect_object

Function

Lists all indexes in a database that are currently offline because of corruption detected on recovery.

Syntax

```
sp_listsuspect_object [dbname]
```

Parameters

dbname – is the name of the database.

Examples

1. `sp_listsuspect_object`

   Lists the suspect indexes in the current database.

2. `sp_listsuspect_object pubs2`

   Lists the suspect indexes in the *pubs2* database.

Comments

- If an index on a data-only-locked table has suspect pages, the entire index is taken offline during recovery. Offline indexes are not considered by the query optimizer.
- Use the system procedure **sp_forceonline_object** to bring an offline index online for repair.
- Indexes on allpages-locked tables are not taken completely offline during recovery; only individual pages of these indexes are taken offline. These pages can be brought online with **sp_forceonline_page**.
- **sp_listsuspect_object** lists the database name, object ID, object name, index ID, and access status for every suspect index in the specified database or, if *dbname* is omitted, in the current user database.
- A value of SA_ONLY in the *access* column means that the index has been forced online for System Administrator use only. A value of BLOCK_ALL means that the index is offline for everyone.
- For more information on recovery fault isolation, see the *System Administration Guide*.

Permissions

Any user can execute **sp_listsuspect_object.**

See Also

System procedures	**sp_forceonline_object**

sp_listsuspect_page

Function

Lists all pages in a database that are currently offline because of corruption detected on recovery.

Syntax

```
sp_listsuspect_page [dbname]
```

Parameters

dbname – is the name of the database.

Examples

1. `sp_listsuspect_page`

   Lists the suspect pages in the current database.

2. `sp_listsuspect_page pubs2`

   Lists the suspect pages in the *pubs2* database.

Comments

- **sp_listsuspect_page** lists the database name, page ID, object, index ID, and access status for every suspect page in the specified database or, if *dbname* is omitted, in the current user database.
- A value of SA_ONLY in the "access" column indicates that the page has been forced online for System Administrator use only. A value of BLOCK_ALL indicates that the page is offline for everyone.

Permissions

Any user can execute **sp_listsuspect_page**.

Tables Used

master.dbo.sysattributes

See Also

System procedures	**sp_forceonline_db, sp_forceonline_page**, **sp_listsuspect_db**, **sp_setsuspect_granularity**, **sp_setsuspect_threshold**

sp_lock

Function

Reports information about processes that currently hold locks.

Syntax

sp_lock [*spid1* [, *spid2*]]

Parameters

spid1 – is the Adaptive Server process ID number from the *master.dbo.sysprocesses* table. Run **sp_who** to get the *spid* of the locking process.

spid2 – is another Adaptive Server process ID number to check for locks.

Examples

1. sp_lock

```
The class column will display the cursor name for locks associated
with a cursor for the current user and the cursor id for other users.
fid spid locktype        table_id  page  dbname  class           context
--- ---- -------------- ---------- ----- ------- --------------  -------
  0   7   Sh_intent       480004741     0 master  Non Cursor Lock NULL
  0  18   Ex_intent        16003088     0 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088   587 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088   590 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088  1114 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088  1140 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088  1283 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088  1362 pubtune Non Cursor Lock NULL
  0  18   Ex_page          16003088  1398 pubtune Non Cursor Lock NULL
  0  18   Ex_page-blk      16003088   634 pubtune Non Cursor Lock NULL
  0  18   Update_page      16003088  1114 pubtune Non Cursor Lock NULL
  0  18   Update_page-blk  16003088   634 pubtune Non Cursor Lock NULL
  0  23   Sh_intent        16003088    0  pubtune Non Cursor Lock NULL
  0  23   Sh_intent       176003658    0  pubtune Non Cursor Lock NULL
  0  23   Ex_intent       208003772    0  pubtune Non Cursor Lock NULL
  1   1   Sh_intent       176003658    0  tpcd   Non Cursor Lock Sync-pt
duration request
  1   1   Sh_intent-blk   208003772    0  tpcd   Non Cursor Lock Sync-pt
```

```
duration request
  1   8   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
  1   9   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
  1  10   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
 11  11   Sh_intent      176003658     0 tpcd   Non Cursor Lock Sync-pt
duration request
 11  12   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
 11  13   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
 11  14   Sh_page         176003658 41571 tpcd    Non Cursor Lock NULL
```

This example shows the lock status of serial processes with *spids* 7, 18, and 23 and two families of processes. The family with *fid* 1 has the coordinating processes with *spid* 1 and worker processes with *spids* 8, 9, and 10. The family with *fid* 11 has the coordinating processes with *spid* 11 and worker processes with *spids* 12, 13, and 14.

2. sp_lock 7

```
The class column will display the cursor name for locks associated
with a cursor for the current user and the cursor id for other users.
fid spid locktype   table_id  page dbname  class             context
--- ---- ---------  --------- ---- ------  ----------------  ----------
 0   7   Sh_intent  480004741    0 master  Non Cursor Lock    NULL
```

Displays information about the locks currently held by *spid 7*.

Comments

- **sp_lock** with no parameters reports information on all processes that currently hold locks.
- The only user control over locking is through the use of the **holdlock** keyword in the **select** statement.
- Use the **object_name** system function to derive a table's name from its ID number.
- **sp_lock** output is ordered by *fid* and then *spid*.
- The *loid* column identifies unique lock owner ID of the blocking transaction**.** Even *loid* values indicate that a local transaction owns the lock. Odd values indicate that an external transaction owns the lock.
- The *locktype* column indicates whether the lock is a shared lock ("Sh" prefix), an exclusive lock ("Ex" prefix) or an update lock, and whether the lock is held on a table ("table" or "intent") or on a page ("page").

  A "blk" suffix in the "locktype" column indicates that this process is blocking another process that needs to acquire a lock.

As soon as this process completes, the other process(es) moves forward. A "demand" suffix in the "locktype" column indicates that the process is attempting to acquire an exclusive lock. For more information about lock types, see the *Performance and Tuning Guide*.

- The *class* column indicates whether a lock is associated with a cursor. It displays one of the following:
  - "Non Cursor Lock" indicates that the lock is not associated with a cursor.
  - "Cursor Id *number*" indicates that the lock is associated with the cursor ID number for that Adaptive Server process ID.
  - A cursor name indicates that the lock is associated with the cursor *cursor_name* that is owned by the current user executing **sp_lock**.
- The *fid* column identifies the family (including the coordinating process and its worker processes) to which a lock belongs. Values for *fid* are:
  - A zero value indicates that the task represented by the *spid* is executed serially. It is not participating in parallel execution.
  - A nonzero value indicates that the task (*spid*) holding the lock is a member of a family of processes (identified by *fid*) executing a statement in parallel. If the value is equal to the *spid*, it indicates that the task is the coordinating process in a family executing a query in parallel.
- The *context* column identifies the context of the lock. Worker processes in the same family have the same context value. Legal values for "context" are as follows:
  - "NULL" means that the task holding this lock is either a query executing serially, or is a query executing in parallel in transaction isolation level 1.
  - "Sync-pt duration request" means that the task holding the lock will hold the lock until the query is complete.

    A lock's context may be "Sync-pt duration request" if the lock is a table lock held as part of a parallel query, if the lock is held by a worker process at transaction isolation level 3, or if the lock is held by a worker process in a parallel query and must be held for the duration of the transaction.
  - "Ind pg" indicates locks on index pages (allpages-locked tables only)

  - "Inf key" indicates an infinity key lock (for certain range queries at transaction isolation level 3 on data-only-locked tables)
  - "Range" indicates a range lock (for range queries at transaction isolation level 3 on data-only-locked tables)

  These new values may appear in combination with "Fam dur" (which replaces "Sync pt duration") and with each other, as applicable.
- The *row* column displays the row number for row-level locks.
- **sp_lock** output also displays the following lock types:
  - "Sh_row" indicates shared row locks
  - "Update_row" indicates update row locks
  - "Ex_row" indicates exclusive row locks

Permissions

Any user can execute **sp_lock**.

Tables Used

master.dbo.spt_values, master.dbo.syslocks, master.dbo.sysobjects

See Also

Commands	**kill**, **select**
System procedures	**sp_familylock**, **sp_who**

sp_locklogin

Function

Locks an Adaptive Server account so that the user cannot log in or displays a list of all locked accounts.

Syntax

```
sp_locklogin [loginame, "{lock | unlock}"]
```

Parameters

loginame – is the name of the account to be locked or unlocked.

lock | **unlock** – specifies whether to lock or unlock the account.

Examples

1. `sp_locklogin charles, "lock"`

   Locks the login account for the user “charles.”

2. `sp_locklogin`

   Displays a list of all locked accounts.

Comments

- Locking an Adaptive Server login account prevents that user from logging in. Use **sp_locklogin** instead of **sp_droplogin** for the following reasons:
  - You cannot drop a login who is a user in any database, and you cannot drop a user from a database if the user owns any objects in that database or has granted any permissions on objects to other users.
  - Adaptive Server may reuse the dropped login account’s server user ID (*suid*) when the next login account is created. This occurs only when the dropped login holds the highest *suid* in *syslogins*; however, it could compromise accountability if execution of **sp_droplogin** is not being audited. In addition, it is possible that the user with the reused *suid* will actually be able to access database objects that were authorized for the old *suid*.
  - You cannot drop the last remaining System Security Officer’s or System Administrator’s login account.
- **sp_locklogin** with no parameters returns a list of all the locked accounts.

- You can lock an account that is currently logged in. The user receives a warning that his or her account has been locked, but is not locked out of the account until he or she logs out.
- A locked account can be specified as a Database Owner and can own objects in any database.
- Locking an account that is already locked or unlocking an unlocked account has no effect.
- When locking a System Security Officer's login account, **sp_locklogin** verifies that at least one other unlocked System Security Officer's account exists. Similarly, **sp_locklogin** verifies that there is always an unlocked System Administrator's account. An attempt to lock the last remaining unlocked System Administrator or System Security Officer account causes **sp_locklogin** to return an error message and fail.

Permissions

Only a System Administrator or a System Security Officers can execute **sp_locklogin**.

Tables Used

master.dbo.sysloginroles, *master.dbo.syslogins*, *master.dbo.sysprocesses*, *sysobjects*

See Also

System procedures	**sp_addlogin**, **sp_modifylogin**, **sp_password**

sp_logdevice

Function

Moves the transaction log of a database with log and data on the same device to a separate database device.

Syntax

sp_logdevice *dbname*, *devname*

Parameters

dbname – is the name of the database whose *syslogs* table, which contains the transaction log, to put on a specific logical device.

devname – is the logical name of the device on which to put the *syslogs* table. This device must be a database device associated with the database (named in **create database** or **alter database**). Run **sp_helpdb** for a report on the database's devices.

Examples

1.
```
create database products on default = 10, logs = 2
go
sp_logdevice products, logs
go
```

   Creates the database *products* and puts the table *products.syslogs* on the database device *logs*.

2.
```
alter database test log on logdev
go
sp_logdevice test, logdev
go
```

   For the database *test* with log and data on the same device, places the log for *test* on the log device *logdev*.

Comments

- The **sp_logdevice** procedure affects only future allocations of space for *syslogs*. This creates a window of vulnerability during which the first pages of your log remain on the same device as your data. Therefore, the preferred method of placing a transaction log on a separate device is the use of the **log on** option to **create database**, which immediately places the entire transaction log on a separate device.

- Place transaction logs on separate database devices, for both recovery and performance reasons.

  A very small, noncritical database could keep its log together with the rest of the database. Such databases use **dump database** to back up the database and log and **dump transaction with truncate_only** to truncate the log.

- **dbcc checkalloc** and **sp_helplog** show some pages for *syslogs* still allocated on the database device until after the next **dump transaction**. After that, the transaction log is completely transferred to the device named when you executed **sp_logdevice**.
- The size of the device required for the transaction log varies, depending on the amount of update activity and the frequency of transaction log dumps. As a rule, allocate to the log device 10 percent to 25 percent of the space you allocate to the database itself.
- Use **sp_logdevice** only for a database with log and data on the same device. Do not use **sp_logdevice** for a database with log and data on separate devices.
- To increase the amount of storage allocated to the transaction log use **alter database**. If you used the **log on** option to **create database** to place a transaction log on a separate device, use:

  sp_extendsegment *segname, devname*

  to increase the size of the log segment. If you did not use **log on**, execute **sp_logdevice**.

  The device or segment on which you put *syslogs* is used **only** for the *syslogs* table. To increase the amount of storage space allocated for the rest of the database, specify any device other than the log device when you issue the **alter database** command.
- Use the **disk init** command to format a new database device for databases or transaction logs.
- For more information, see the *System Administration Guide.*

Permissions

Only the Database Owner or a System Administrator can execute **sp_logdevice**.

Tables Used

master.dbo.sysdatabases, master.dbo.sysdevices, master.dbo.sysusages, sysobjects

See Also

Commands	**alter database**, **create database**, **dbcc**, **disk init**, **dump database**, **dump transaction**, **select**
System procedures	**sp_extendsegment**, **sp_helpdevice**

sp_loginconfig

(Windows NT only)

Function

Displays the value of one or all integrated security parameters.

Syntax

```
sp_loginconfig ["parameter_name"]
```

Parameters

parameter_name – is the name of the integrated security parameter you want to examine. Values are: **login mode**, **default account**, **default domain**, **set host**, **key _**, **key $**, **key @**, and **key #**.

Examples

1. sp_loginconfig

```
name                 config_item
-------------------- --------------------
login mode           standard
default account      NULL
default domain       NULL
set host             false
key _                domain separator
key $                space
key @                space
key #                -
```

Displays the values of all integrated security parameters.

2. sp_loginconfig "login mode"

```
name                 config_item
-------------------- --------------------
login mode           standard
```

Displays the value of the **login mode** security parameter.

Comments

- The values of integrated security parameters are stored in the Windows NT Registry. See the chapter on login security in *Configuring Adaptive Server for Windows NT* for instructions on changing the parameters.

- **sp_loginconfig** displays the *config_item* values that were in effect when you started Adaptive Server. If you changed the Registry values after starting Adaptive Server, those values are not reflected in the **sp_loginconfig** output.

Permissions

Only a System Administrator can execute **sp_loginconfig**.

Tables Used

sysobjects

See Also

System procedures	**sp_revokelogin**

sp_logininfo

(Windows NT only)

Function

Displays all roles granted to Windows NT users and groups with **sp_grantlogin**.

Syntax

```
sp_logininfo ["login_name" | "group_name"]
```

Parameters

login_name – is the network login name of the Windows NT user.

group_name – is the Windows NT group name.

Examples

1. sp_logininfo regularjoe

```
account name    mapped login name  type            privilege
--------------- ------------------ --------------- --------------
HAZE\regularjoe HAZE_regularjoe    user            'oper_role'
```

Displays the permissions granted to the Windows NT user "regularjoe."

2. sp_logininfo

```
account name                mapped login name
    type
        privilege
---------------------       ------------------
     ----------------------------
        ------------------------------------------------------------
BUILTIN\Administrators      BUILTIN\Administrators
    group
        'sa_role sso_role oper_role sybase_ts_role navigator_role
              replication_role'
HAZE\regularjoe             HAZE_regularjoe
    user
        'oper_role'
PCSRE\randy                 PCSRE_alexander
    user
        'default'
```

Displays all permissions that were granted to Windows NT users and groups with **sp_grantlogin**.

Comments

- **sp_logininfo** displays all roles granted to Windows NT users and groups with **sp_grantlogin**.
- You can omit the domain name and domain separator (\) when specifying the Windows NT user name or group name.

Permissions

Only a System Administrator can execute **sp_logininfo**.

Tables Used

sysobjects

See Also

Commands	**grant**, **setuser**
System procedures	**sp_displaylogin**, **sp_grantlogin**, **sp_revokelogin**, **sp_role**, **sp_who**

sp_logiosize

Function

Changes the log I/O size used by Adaptive Server to a different memory pool when doing I/O for the transaction log of the current database.

Syntax

```
sp_logiosize ["default" | "size" | "all"]
```

Parameters

default – sets the log I/O size for the current database to Adaptive Server's default value (4K), if a 4K memory pool is available in the cache. Otherwise, Adaptive Server sets the log I/O size to 2K. Since **default** is a keyword, the quotes are required when specifying this parameter.

size – is the size to set the log I/O for the current database. Values are **2**, **4**, **8**, and **16**. You must enclose the value in quotes.

all – displays the log I/O size configured for all databases grouped by the cache name.

Examples

1. sp_logiosize

```
The transaction log for database 'master' will use
I/O size of 2 Kbytes.
```

Displays the log I/O size configured for the current database.

2. sp_logiosize "8"

Changes the log I/O size of the current database to use the 8K memory pool. If the database's transaction log is bound to a cache that does not have an 8K memory pool, Adaptive Server returns an error message indicating that such a pool does not exist, and the current log I/O size does not change.

3. sp_logiosize "default"

Changes the log I/O size of the current database to Adaptive Server's default value (4K). If a 4K memory pool does not exist in the cache used by the transaction log, Adaptive Server uses the 2K memory pool.

4. sp_logiosize "all"

```
Cache name: default data cache
Data base                      Log I/O Size
-----------------------------  ------------
master                         2 Kb
tempdb                         2 Kb
model                          2 Kb
sybsystemprocs                 2 Kb
pubs3                          2 Kb
pubtune                        2 Kb
dbccdb                         2 Kb
sybsyntax                      2 Kb
```

Displays the log I/O size configured for all databases.

Comments

- **sp_logiosize** displays or changes the log I/O size for the current database. Any user can execute **sp_logiosize** to display the configured log I/O size. Only a System Administrator can change the log I/O size.
- If you specify **sp_logiosize** with no parameters, Adaptive Server displays the log I/O size of the current database.
- When you change the log I/O size, it takes effect immediately. Adaptive Server records the new I/O size for the database in the *sysattributes* table.
- Any value you specify for **sp_logiosize** must correspond to an existing memory pool configured for the cache used by the database's transaction log. Specify these pools using the **sp_poolconfig** system procedure.

  Adaptive Server defines the default log I/O size of a database as 4K, if a 4K memory pool is available in the cache. Otherwise, Adaptive Server sets the log I/O size to 2K (a 2K memory pool is always present in any cache). For most work loads, a log I/O size of 4K performs much better than one of 2K, so each cache used by a transaction log should have a 4K memory pool. For more information about configuring caches and memory pools, see the *System Administration Guide* and the *Performance and Tuning Guide*.
- If the transaction logs for one or more databases are bound to a cache of type **logonly**, any memory pools in that cache that have I/O sizes larger than the log I/O size defined for those databases will **not** be used.

For example, assume that only two databases have their transaction logs bound to a "log only" cache containing 2K, 4K, and 8K memory pools. By default, **sp_logiosize** sets the log I/O size for these parameters at 4K, and the 8K pool is not used. Therefore, to avoid wasting cache space, be cautious when configuring the log I/O size.

- During recovery, only the 2K memory pool of the default cache is active, regardless of the log I/O size configured for a database. Transactions logs are read into the 2K pool of the default cache, and all transactions that must be rolled back, or rolled forward, read data pages into the default data cache.

Permissions

Only a System Administrator can execute **sp_logiosize** to change the log I/O size for the current database. Any user can execute **sp_logiosize** to display the log I/O size values.

Tables Used

sysattributes

See Also

System procedures	**sp_cacheconfig**, **sp_poolconfig**

sp_modifylogin

Function

Modifies the default database, default language, default role activation, or full name for an Adaptive Server login account.

Syntax

sp_modifylogin *account*, *column*, *value*

Parameters

account – is the login account to be modified.

column – specifies the name of the option to be changed. The options are:

Option	Definition
defdb	The "home" database to which the user is connected when he or she logs in.
deflanguage	The official name of the user's default language.
fullname	The user's full name.
"add default role"	The role or roles to be activated by default at login.
"drop default role"	The role or roles to be dropped from the list of roles activated by default at login.

value – is the new value for the specified option.

Examples

1. **sp_modifylogin sarah, defdb, "pubs2"**

   Changes the default database for "sarah" to *pubs2*.

2. **sp_modifylogin claire, deflanguage, "french"**

   Sets the default language for "claire" to French.

3. **sp_modifylogin clemens, fullname, "Samuel Clemens"**

   Changes the full name of user "clemens" to "Samuel Clemens."

4. **sp_modifylogin csmith, "add default role", specialist_role**

   Adds the *specialist* role to the list of roles activated by default when user csmith logs in.

5. **sp_modifylogin hpillai, "drop default role", intern_role**

   Drops the *intern* role from the list of roles activated by default when user "hpillai" logs in.

Comments

- Set a default database, language, or full name either with **sp_modifylogin** or with **sp_addlogin** when first adding the user's login to Adaptive Server.
  - If you do not specify a default database, the user's default is *master*.
  - If you do not specify a language, the user's default language is set to the server's default language.
  - If you do not specify a full name, that column in *syslogins* remains blank.

Changing a User's Default Database

- After **sp_modifylogin** is executed to change the user's default database, the user is connected to the new *defdb* the next time he or she logs in. However, the user cannot access the database until the Database Owner gives the user access through **sp_adduser** or **sp_addalias**, or unless there is a "guest" user in the database's *sysusers* table. If the user does not have access to the database by any of these means, she or he is connected to *master* and an error message appears.
- If a user's default database is dropped, or if the user is dropped from the database, the user is connected to *master* on his or her next login, and an error message appears.
- If a user's default language is dropped from the server, the server-wide default language is used as the initial language setting, and a message appears.

Changing a User's Role Activation

- Use **sp_modifylogin** to set a role to be activated by default at login or to drop a role from those activated by default at login.

Permissions

Only a System Administrator can execute **sp_modifylogin** to change the default database, default language, or full name of another user. Only a System Security Officer can execute **sp_modifylogin** to activate

another user's roles by default at login. Any user can execute **sp_modifylogin** to change his or her own login account.

Tables Used

master..syslogins, sysobjects, sysloginroles

See Also

System procedures	**sp_activeroles**, **sp_addlogin**, **sp_displaylogin**, **sp_displayroles**, **sp_helprotect**

sp_modify_resource_limit

Function

Changes a resource limit by specifying a new limit value, or the action to take when the limit is exceeded, or both.

Syntax

```
sp_modify_resource_limit {name, appname }
   , rangename , limittype , limitvalue , enforced
   , action , scope
```

Parameters

name – is the Adaptive Server login to which the limit applies. You must specify either a *name* or an *appname* or both. To modify a limit that applies to all users of a particular application, specify a *name* of **null**.

appname – is the name of the application to which the limit applies. You must specify either a *name* or an *appname* or both. If the limit applies to all applications used by *name*, specify an *appnam*e of **null**. If the limit governs a particular application, specify the application name that the client program passes to the Adaptive Server in the login packet.

rangename – is the time range during which the limit is enforced. You cannot modify this value, but you must specify a non-null value to uniquely identify the resource limit.

limittype – is the type of resource to which the limit applies. You cannot modify this value, but you must specify a non-null value to uniquely identify the resource limit. The value must be one of the following:

Limit Type	Description
row_count	Limits the number of rows a query can return
elapsed_time	Limits the number of seconds in wall-clock time that a query batch or transaction can run
io_cost	Limits either the actual cost, or the optimizer's cost estimate, for processing a query

limit_value – is the maximum amount of the server resource that the login or application can use before Adaptive Server enforces the limit. This must be a positive integer less than or equal to 2^{31} or **null** to retain the existing value. The following table indicates what value to specify for each limit type:

Limit Type	Limit Value
row_count	The maximum number of rows a query can return before the limit is enforced
elapsed_time	The maximum number of seconds in wall-clock time that a query batch or transaction can run before the limit is enforced
io_cost	A unitless measure derived from optimizer's costing formula

enforced – determines whether the limit is enforced prior to or during query execution. You cannot modify this value. Use **null** as a placeholder.

action – is the action to take when the limit is exceeded. The following codes apply to all limit types:

Action Code	Description
1	Issues a warning
2	Aborts the query batch
3	Aborts the transaction
4	Kills the session
null	Retains the existing value

scope – is the scope of the limit. You cannot modify this value. You can use **null** as a placeholder.

Examples

1.
```
sp_modify_resource_limit robin, NULL, weekends,
row_count, 3000, NULL, 1, NULL
```

Modifies a resource limit that applies to all applications used by "robin" during the *weekends* time range. The limit issues a warning when a query is expected to return more than 3000 rows.

```
2. sp_modify_resource_limit NULL, acctg,
   "at all times", elapsed_time, 45, 2, 2, 6
```

Modifies a resource limit that applies to the *acctg* application on all days of the week and at all times of the day. The limit aborts the query batch when estimated query processing time exceeds 45 seconds.

Comments

- You cannot change the login or application to which a limit applies or specify a new time range, limit type, enforcement time, or scope.
- The modification of a resource limit causes the limits for each session for that login and / or application to be rebound at the beginning of the next query batch for that session.
- For more information, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_modify_resource_limit.**

Tables Used

master..sysresourcelimits, master..systimeranges, master..spt_limit_types

See Also

System procedures	**sp_add_resource_limit**, **sp_drop_resource_limit**, **sp_help_resource_limit**

sp_modify_time_range

Function

Changes the start day, start time, end day, and / or end time associated with a named time range.

Syntax

```
sp_modify_time_range name, startday, endday,
   starttime, endtime
```

Parameters

name – is the name of the time range. This must be the name of a time range stored in the *systimeranges* system table of the *master* database.

startday – is the day of the week on which the time range begins. This must be the full weekday name for the default server language, as stored in the *syslanguages* system table of the *master* database, or **null** to keep the existing *startday*.

endday – is the day of the week on which the time range ends. This must be the full weekday name for the default server language, as stored in the *syslanguages* system table of the *master* database, or **null** to keep the existing end day. The *endday* can fall either earlier or later in the week than the *startday*, or it can be the same day as the *startday*.

starttime – is time of day at which the time range begins. Specify the *starttime* in terms of a twenty-four hour clock, with a value between 00:00 and 23:59. Use the following form:

"*HH*:*MM*"

or **null** to keep the existing *starttime*.

endtime – is the time of day at which the time range ends. Specify the *endtime* in terms of a twenty-four hour clock, with a value between 00:00 (midnight) and 23:59. Use the following form:

"*HH*:*MM*"

or **null** to keep the existing *endtime*. The *endtime* must occur later in the day than the *starttime*, unless *endtime* is 00:00.

➤ ***Note***

For time ranges that span the entire day, specify a start time of "00:00" and an end time of "23:59".

Examples

1. **`sp_modify_time_range business_hours, NULL, Saturday, NULL, NULL`**

   Changes the end day of the *business_hours* time range from Friday to Saturday. Retains the existing start day, start time, and end time.

2. **`sp_modify_time_range before_hours, Monday, Saturday, NULL, "08:00"`**

   Specifies a new end day and end time for the *before_hours* time range.

Comments

- You cannot modify the "at all times" time range.
- It is possible to modify a time range so that it overlaps with one or more other time ranges.
- The modification of time ranges through the system stored procedures does not affect the active time ranges for sessions currently in progress.
- Changes to a resource limit that has a transaction as its scope does not affect any transactions currently in progress.
- For more information, see the *System Administration Guide*.

Permissions

Only a System Administrator can execute **sp_modify_time_range.**

Tables Used

master..systimeranges, master..syslanguages

See Also

System procedures	**sp_add_resource_limit**, **sp_add_time_range**, **sp_drop_time_range**

sp_modifythreshold

Function

Modifies a threshold by associating it with a different threshold procedure, free-space level, or segment name. You **cannot** use **sp_modifythreshold** to change the amount of free space or the segment name for the last-chance threshold.

Syntax

```
sp_modifythreshold dbname, segname, free_space
   [, new_proc_name] [, new_free_space]
   [, new_segname]
```

Parameters

dbname – is the database for which to change the threshold. This must be the name of the current database.

segname – is the segment for which to monitor free space. Use quotes when specifying the "default" segment.

free_space – is the number of free pages at which the threshold is crossed. When free space in the segment falls below this level, Adaptive Server executes the associated stored procedure.

new_proc_name – is the new stored procedure to execute when the threshold is crossed. The procedure can be located in any database on the current Adaptive Server or on an Open Server. Thresholds cannot execute procedures on remote Adaptive Servers.

new_free_space – is the new number of free pages to associate with the threshold. When free space in the segment falls below this level, Adaptive Server executes the associated stored procedure.

new_segname – is the new segment for which to monitor free space. Use quotes when specifying the "default" segment.

Examples

1. `sp_modifythreshold mydb, "default", 200, NULL, 175`

Modifies a threshold on the "default" segment of the *mydb* database to execute when free space on the segment falls below 175 pages instead of 200 pages. NULL is a placeholder indicating that the procedure name is not being changed.

2. `sp_modifythreshold mydb, data_seg, 250, new_proc`

Modifies a threshold on the *data_seg* segment of *mydb* so that it executes the *new_proc* procedure.

Comments

- For more information, see the *System Administration Guide.*

Crossing a Threshold

- When a threshold is crossed, Adaptive Server executes the associated stored procedure. Adaptive Server uses the following search path for the threshold procedure:
  - If the procedure name does not specify a database, Adaptive Server looks in the database in which the threshold was crossed.
  - If the procedure is not found in this database and the procedure name begins with "sp_", Adaptive Server looks in the *sybsystemprocs* database.

  If the procedure is not found in either database, Adaptive Server sends an error message to the error log.
- Adaptive Server uses a **hysteresis value**, the global variable *@@thresh_hysteresis*, to determine how sensitive thresholds are to variations in free space. Once a threshold executes its procedure, it is deactivated. The threshold remains inactive until the amount of free space in the segment rises to *@@thresh_hysteresis* pages above the threshold. This prevents thresholds from executing their procedures repeatedly in response to minor fluctuations in free space.

The Last-Chance Threshold

- By default, Adaptive Server monitors the free space on the segment where the log resides and executes **sp_thresholdaction** when the amount of free space is less than that required to permit a successful dump of the transaction log. This amount of free space, the **last-chance threshold**, is calculated by Adaptive Server and cannot be changed by users.
- If the last-chance threshold is crossed before a transaction is logged, Adaptive Server suspends the transaction until log space is freed. Use **sp_dboption** to change this behavior for a particular database. Setting the **abort tran on log full** option to **true** causes

Adaptive Server to roll back all transactions that have not yet been logged when the last-chance threshold is crossed.

- You cannot use **sp_modifythreshold** to change the free-space value or segment name associated with the last-chance threshold.

Other Thresholds

- Each database can have up to 256 thresholds, including the last-chance threshold.
- Each threshold must be at least 2 times *@@thresh_hysteresis* pages from the next closest threshold.
- Use **sp_helpthreshold** for information about existing thresholds.
- Use **sp_dropthreshold** to drop a threshold from a segment.

Creating Threshold Procedures

- Any user with **create procedure** permission can create a threshold procedure in a database. Usually, a System Administrator creates **sp_thresholdaction** in the *master* database, and Database Owners create threshold procedures in user databases.
- **sp_modifythreshold** does not verify that the specified procedure exists. It is possible to associate a threshold with a procedure that does not yet exist.
- **sp_modifythreshold** checks to ensure that the user modifying the threshold procedure has been directly granted the "sa_role". All system roles active when the threshold procedure is modified are entered in *systhresholds* as valid roles for the user writing the procedure. However, only directly granted system roles are activated when the threshold fires. Indirectly granted system roles and user-defined roles are not activated.
- Adaptive Server passes four parameters to a threshold procedure:
  - *@dbname*, *varchar*(30), which identifies the database
  - *@segment_name*, *varchar*(30), which identifies the segment
  - *@space_left*, *int*, which indicates the number of free pages associated with the threshold
  - *@status*, *int*, which has a value of 1 for last-chance thresholds and 0 for other thresholds

  These parameters are passed by position rather than by name; your threshold procedure can use other names for them, but the

procedure must declare them in the order shown and with the correct datatypes.

- It is not necessary to create a different procedure for each threshold. To minimize maintenance, create a single threshold procedure in the *sybsystemprocs* database that can be executed by all thresholds.
- Include **print** and **raiserror** statements in the threshold procedure to send output to the error log.

Executing Threshold Procedures

- Tasks that are initiated when a threshold is crossed execute as background tasks. These tasks do not have an associated terminal or user session. If you execute **sp_who** while these tasks are running, the *status* column shows "background".
- Adaptive Server executes the threshold procedure with the permissions of the user who modified the threshold, at the time he or she executed **sp_modifythreshold**, minus any permissions that have since been revoked.
- Each threshold procedure uses one user connection, for as long as it takes to execute the procedure.

Disabling Free-Space Accounting

- Use the **no free space acctg** option of **sp_dboption** to disable free-space accounting on non-log segments.
- You cannot disable free-space accounting on log segments.

◆ ***WARNING!***

System procedures cannot provide accurate information about space allocation when free-space accounting is disabled.

Creating Last-Chance Thresholds for Pre-Release 10.0 Databases

- When you upgrade a pre-release 10.0 database to the current release, it does not automatically acquire a last-chance threshold. Use the **lct_admin** system function to create a last-chance threshold in an existing database.
- Only databases that store their logs on a separate segment can have a last-chance threshold. Use **sp_logdevice** to move the transaction log to a separate device.

Permissions

Only the Database Owner or a System Administrator can execute **sp_modifythreshold.**

Tables Used

master..sysusages, sysobjects, syssegments, systhresholds

See Also

Commands	**create procedure**, **dump transaction**
System procedures	**sp_addthreshold**, **sp_dboption**, **sp_dropthreshold**, **sp_helpthreshold**, **sp_thresholdaction**

sp_monitor

Function

Displays statistics about Adaptive Server.

Syntax

sp_monitor

Parameters

None.

Examples

1. sp_monitor

```
last_run               current_run          seconds
-------------------    -------------------  ---------
Jan 29 1987 10:11AM    Jan 29 1987 10:17AM  314

cpu_busy           io_busy     idle
---------------    ---------   --------------
4250(215)-68%      67(1)-0%    109(100)-31%

packets_received       packets_sent    packet_errors
----------------       ------------    ------------
781(15)                10110(9596)     0(0)

total_read     total_write total_errors    connections
-----------    ------------------------    -----------
394(67)        5392(53)    0(0)            15(1)
```

Reports information about how busy Adaptive Server has been.

Comments

- Adaptive Server keeps track of how much work it has done in a series of global variables. **sp_monitor** displays the current values of these global variables and how much they have changed since the last time the procedure executed.
- For each column, the statistic appears in the form *number(number)-number%* or *number(number)*.
  - The first number refers to the number of seconds (for "cpu_busy", "io_busy", and "idle") or the total number (for the other columns) since Adaptive Server restarted.
  - The number in parentheses refers to the number of seconds or the total number since the last time **sp_monitor** was run. The

percent sign indicates the percentage of time since **sp_monitor** was last run.

For example, if the report shows "cpu_busy" as "4250(215)-68%", it means that the CPU has been busy for 4250 seconds since Adaptive Server was last started, 215 seconds since **sp_monitor** last ran, and 68 percent of the total time since **sp_monitor** was last run.

For the "total_read" column, the value 394(67) means there have been 394 disk reads since Adaptive Server was last started, 67 of them since the last time **sp_monitor** was run.

- Table 7-19 describes the columns in the **sp_monitor** report, the equivalent global variables, if any, and their meanings. With the exception of "last_run", "current_run" and "seconds", these column headings are also the names of global variables—except that all global variables are preceded by @@. There is also a difference in the units of the numbers reported by the global variables—the numbers reported by the global variables are not milliseconds of CPU time, but machine ticks.

Table 7-19: Columns in the sp_monitor report

Column Heading	Equivalent Variable	Meaning
last_run		Clock time at which the **sp_monitor** procedure last ran.
current_run		Current clock time.
seconds		Number of seconds since **sp_monitor** last ran.
cpu_busy	*@@cpu_busy*	Number of seconds in CPU time that Adaptive Server's CPU was doing Adaptive Server work.
io_busy	*@@io_busy*	Number of seconds in CPU time that Adaptive Server has spent doing input and output operations.
idle	*@@idle*	Number of seconds in CPU time that Adaptive Server has been idle.
packets_received	*@@pack_received*	Number of input packets read by Adaptive Server.
packets_sent	*@@pack_sent*	Number of output packets written by Adaptive Server.
packet_errors	*@@packet_errors*	Number of errors detected by Adaptive Server while reading and writing packets.
total_read	*@@total_read*	Number of disk reads by Adaptive Server.
total_write	*@@total_write*	Number of disk writes by Adaptive Server.

Table 7-19: Columns in the sp_monitor report (continued)

Column Heading	Equivalent Variable	Meaning
total_errors	*@@total_errors*	Number of errors detected by Adaptive Server while reading and writing.
connections	*@@connections*	Number of logins or attempted logins to Adaptive Server.

- The first time **sp_monitor** runs after Adaptive Server start-up, the number in parentheses is meaningless.
- Adaptive Server's housekeeper task uses the server's idle cycles to write changed pages from cache to disk. This process affects the values of the "cpu_busy", "io_busy", and "idle" columns reported by **sp_monitor.** To disable the housekeeper task and eliminate these effects, set the **housekeeper free write percent** configuration parameter to 0:

  ```
  sp_configure "housekeeper free write percent", 0
  ```

Permissions

Only a System Administrator can execute **sp_monitor.**

Tables Used

master.dbo.sysengines, *master.dbo.spt_monitor*, *sysobjects*

See Also

System procedures	**sp_who**

sp_monitorconfig

Function

Displays cache usage statistics regarding metadata descriptors for indexes, objects, and databases. **sp_monitorconfig** also reports statistics on auxiliary scan descriptors used for referential integrity queries, and usage statistics for transaction descriptors and DTX participants.

Syntax

```
sp_monitorconfig "configname"
```

Parameters

configname – is all or part of the configuration parameter name whose monitoring information is being queried. Valid configuration parameters are **open indexes**, **open objects**, **open databases**, **aux scan descriptors**, **txn to pss ratio**, **number of dtx participants**, and **all**. Specifying **all** displays descriptor help information for all indexes, objects, databases, and auxiliary scan descriptors in the server.

Examples

1. sp_monitorconfig "open"

```
Configuration option is not unique.
option_name                    config_value run_value
------------------------------ ------------ -----------
curread change w/ open cursors            1           1
number of open databases                 12          12
number of open indexes                  500         500
number of open objects                  500         500
open index hash spinlock ratio          100         100
open index spinlock ratio               100         100
open object spinlock ratio              100         100
```

2. sp_monitorconfig "open objects"

```
Usage information at date and time: Jan 14 1997 8:54AM.

Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
number of open  217       283      56.60      300               No
objects
```

In this example, there are 283 active object metadata descriptors, with 217 free. The maximum used at a peak period since Adaptive Server was last started is 300. You can then reset the

size to 330, for example, to accommodate the 300 maximum used metadata descriptors, plus space for 10 percent more:

```
sp_configure "number of open objects", 330
```

3. **sp_monitorconfig "open indexes"**

```
Usage information at date and time: Jan 14 1997 8:55AM.

Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
number of open  556       44       7.33       44                No
indexes
```

In this example, the maximum number of index metadata descriptors is 44. You can reset the size to 100, the minimum acceptable value:

```
sp_configure "number of open indexes", 100
```

4. **sp_monitorconfig "aux scan descriptors"**

```
Usage information at date and time: Jan 14 1997 8:56AM.

Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
number of aux   170       30       15.00      32                NA
scan
descriptors
```

In this example, the number of active scan descriptors is 30, though Adaptive Server is configured to use 200. Use the **number of aux scan descriptors** configuration parameter to reset the value to at least 32. A safe setting is 36, to accommodate the 32 scan descriptors, plus space for 10 percent more.

5. **sp_monitorconfig "number of open databases"**

```
Usage information at date and time: Jan 14 1997 8:57AM.

Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
number of open  0         5        100.00     5                 Yes
databases
```

In this example, Adaptive Server is configured for 5 open databases, all of which have been used in the current session. However, as indicated by the "Re-used" column, an additional database needs to be opened. If all 5 databases are in use, an error may result, unless the descriptor for a database that is not in use can be reused. To prevent an error, reset **number of open databases** to a higher value.

6. **sp_monitorconfig "txn to pss ratio"**

```
Usage information at date and time: Jun 18 1999 8:54AM.
```

```
Name            # Free   # Active   % Active  # Max Ever Used   Re-used
--------------  --------  ------   --------   ---------------   -------
txn to pss       784       80       10.20        523              NA
ratio
```

In this example, only 10.2 percent of the transaction descriptors are currently being used. However, the maximum number of transaction descriptors used at a peak period since Adaptive Server was last started is 523.

Comments

- **sp_monitorconfig** displays cache usage statistics regarding metadata descriptors for indexes, objects, and databases, such as the number of metadata descriptors currently in use by the server.
- **sp_monitorconfig** also reports the number of auxiliary scan descriptors in use. A scan descriptor manages a single scan of a table when queries are run on the table.
- The columns in the **sp_monitorconfig** output provide the following information:
  - "# Free" specifies the number of available metadata or auxiliary scan descriptors not currently used.
  - "# Active" specifies the number of metadata or auxiliary scan descriptors installed in cache (that is, active).
  - "% Active" specifies the percentage of cached or active metadata or auxiliary scan descriptors.
  - "# Max Ever Used" specifies the maximum number of metadata or auxiliary scan descriptors that have been in use since the server was started.
  - "Re-used" specifies whether a metadata descriptor was reused in order to accommodate an increase in indexes, objects, or databases in the server. The returned value is "Yes", "No" or "NA" (for configuration parameters that do not support the reuse mechanism, such as the number of **aux scan descriptors**).
- Use the value in the "# Max Ever Used" column as a basis for determining an appropriate number of descriptors; be sure to add about 10 percent for the final setting. For example, if the maximum number of index metadata descriptors used is 142, you might set the **number of open indexes** configuration parameter to 157.
- If the "Re-used" column states "Yes," reset the configuration parameter to a higher value. When descriptors need to be reused, there can be performance problems, particularly with open

databases. An open database contains a substantial amount of metadata information, which means that to fill up an open database, Adaptive Server needs to access the metadata on the disk many times; the server can also have a spinlock contention problem. To check for spinlock contention, use the system procedure **sp_sysmon**. For more information, see the *Performance and Tuning Guide*. To find the current number of indexes, objects, or databases, use **sp_countmetadata**.

- To get an accurate reading, run **sp_monitorconfig** during a normal Adaptive Server peak time period. You can run **sp_monitorconfig** several times during the peak period to ensure that you are actually finding the maximum number of descriptors used.

Permissions

Only a System Administrator can execute **sp_monitorconfig**.

Tables Used

master..sysindexes, sysobjects, sysdatabases

See Also

System procedures	**sp_configure**, **sp_countmetadata**, **sp_helpconfig**, **sp_helpconstraint**

sp_object_stats

Function

Prints lock contention, lock wait-time, and deadlock statistics for tables and indexes.

Syntax

```
sp_object_stats interval [, top_n
   [, dbname, objname [, rpt_option ]]]
```

Parameters

interval – specifies the time period for the sample. It must be in HH:MM:SS form, for example “00:20:00”.

top_n – the number of objects to report, in order of contention. The default is 10.

dbname – the name of the database to report on. If no database name is given, contention on objects in all databases is reported.

objname – the name of a table to report on. If a table name is specified, the database name must also be specified.

rpt_option – must be either **rpt_locks** or **rpt_objlist**.

Examples

1. `sp_object_stats "00:20:00"`

   Reports lock statistics on the top 10 objects server-wide.

2. `sp_object_stats "00:20:00", 5, pubtune`

   Reports only on tables in the *pubtune* database, and lists the five tables that experienced the highest contention.

3. `sp_object_stats "00:15:00", @rpt_option = "rpt_objlist"`

   Prints only the names of the tables that had the highest locking activity, even if contention and deadlocking does not take place.

Comments

- **sp_object_stats** reports on the shared, update, and exclusive locks acquired on tables during a specified sample period. The following reports shows the *titles* tables:

```
Object Name: pubtune..titles (dbid=7, objid=208003772,lockscheme=Datapages)

  Page Locks     SH_PAGE               UP_PAGE              EX_PAGE$
  ----------    ----------            ----------           ----------
  Grants:            94488                 4052                 4828
  Waits:               532                  500                  776
  Deadlocks:             4                    0                   24
  Wait-time:      20603764 ms           14265708 ms          2831556 ms
  Contention:         0.56%                10.98%               13.79%

 *** Consider altering pubtune..titles to Datarows locking.
```

- Table 7-20 shows the meaning of the values.

Table 7-20: Output of sp_object_stats

Output Row	Value
Grants	The number of times the lock was granted immediately.
Waits	The number of times the task needing a lock had to wait.
Deadlocks	The number of deadlocks that occurred.
Wait-times	The total number of milliseconds that all tasks spent waiting for a lock.
Contention	The percentage of times that a task had to wait or encountered a deadlock.

- **sp_object_stats** recommends changing the locking scheme when total contention on a table is more than 15 percent, as follows:
  - If the table uses allpages locking, it recommends changing to datapages locking
  - If the table uses datapages locking, it recommends changing to datarows locking.
- *rpt_option* specifies the report type:
  - **rpt_locks** reports grants, waits, deadlocks and wait times for the tables with the highest contention. **rpt_locks** is the default.
  - **rpt_objlist** reports only the names of the objects that had the highest level of lock activity
- **sp_object_stats** creates a table named *tempdb..syslkstats*. This table is not dropped when the stored procedure completes so that it can be queried by a System Administrator.

- Only one user at a time should execute **sp_object_stats**. If more than one user tries to run **sp_object_stats** simultaneously, the second command may be blocked, or the results may be invalid.
- The *tempdb..syslkstats* table is dropped and re-created each time **sp_object_stats** is executed.
- The structure of *tempdb..syslkstats* is described in Table 7-21.

Table 7-21: Columns in the tempdb..syslkstats table

Column Name	Datatype	Description
dbid	*smallint*	Database ID
objid	*int*	Object ID
lockscheme	*smallint*	Integer values 1–3: Allpages = 1, Datapages = 2, Datarows = 3
page_type	*smallint*	Data page = 0, or index page = 1
stat_name	*char(30)*	The statistics represented by this row
stat_value	*float*	The number of grants, waits or deadlocks, or the total wait time

The values in the *stat_name* column are composed of three parts:

- The first part is “ex” for exclusive lock, “sh” for shared lock, or “up” for update lock.
- The second part is “pg” for page locks, or “row” for row locks.
- The third part is “grants” for locks granted immediately, “waits” for locks that had to wait for other locks to be released, “deadlocks” for deadlocks, and “waittime” for the time waited to acquire the lock.

- If you specify a table name, **sp_object_stats** displays all tables by that name. If more than one user owns a table with the specified name, output for these tables displays the object ID, but not the owner name.

Permission

Only a System Administrator can execute **sp_object_stats**.

Tables Used

Creates the table *tempdb..syslkstats*. This table is not dropped at the end of execution and can be queried via Transact-SQL.

See Also

Commands	**alter table**

sp_passthru

(Component Integration Services only)

Function

Allows the user to pass a SQL command buffer to a remote server.

Syntax

```
sp_passthru server, command, errcode, errmsg, rowcount
    [, arg1, arg2, ... argn]
```

Parameters

server – is the name of a remote server to which the SQL command buffer will be passed. The class of this server must be a supported, non-local server class.

command – is the SQL command buffer. It can hold up to 255 characters.

errcode – is the error code returned by the remote server, if any. If no error occurred at the remote server, the value returned is 0.

errmsg – is the error message returned by the remote server. It can hold up to 255 characters. This parameter is set only if *errcode* is a nonzero number; otherwise NULL is returned.

rowcount – is the number of rows affected by the last command in the command buffer. If the command was an **insert**, **delete**, or **update**, this value represents the number of rows affected even though none were returned. If the last command was a query, this value represents the number of rows returned from the external server.

arg1 ... argn – receives the results from the last row returned by the last command in the command buffer. You can specify up to 250 *arg* parameters. All must be declared as output parameters.

Examples

1.
```
sp_passthru ORACLE, "select date from dual",
@errcode output, @errmsg output, @rowcount output,
@oradate output
```

   Returns the date from the Oracle server in the output parameter *@oradate*. If an Oracle error occurs, the error code is placed in *@errcode* and the corresponding message is placed in *@errmsg*. The *@rowcount* parameter will be set to 1.

Comments

- **sp_passthru** allows the user to pass a SQL command buffer to a remote server. The syntax of the SQL statement or statements being passed is assumed to be the syntax native to the class of server receiving the buffer. No translation or interpretation is performed. Results from the remote server are optionally placed in output parameters.

  Use **sp_passthru** only when Component Integration Services is installed and configured.

- You can include multiple commands in the command buffer. For some server classes, the commands must be separated by semicolons. Refer to the *Component Integration Services User's Guide* for a more complete discussion of query buffer handling in passthru mode.

Return Parameters

- The output parameters *arg1* ... *argn* will be set to the values of corresponding columns from the last row returned by the last command in the command buffer. The position of the parameter determines which column's value the parameter will contain. *arg1* receives values from column 1, *arg2* receives values from column 2, and so on.
- If there are fewer optional parameters than there are returned columns, the excess columns are ignored. If there are more parameters than columns, the remaining parameters are set to NULL.
- An attempt is made to convert each column to the datatype of the output parameter. If the datatypes are similar enough to permit **implicit** conversion, the attempt will succeed. For information on implicit conversion, see "Datatype Conversion Functions" in Chapter 2, "Transact-SQL Functions". For information on which datatype represents the datatypes from each server class when in passthru mode, see the *Component Integration Services User's Guide*.

Permissions

Any user can execute **sp_passthru**.

Tables Used

sysservers, *sysremotelogins*

See Also

System procedures	**sp_autoconnect**, **sp_remotesql**

sp_password

Function

Adds or changes a password for an Adaptive Server login account.

Syntax

sp_password ***caller_passwd, new_passwd*** **[,** ***loginame*****]**

Parameters

caller_passwd – is your password. When you are changing your own password, this is your old password. When a System Security Officer is using **sp_password** to change another user's password, *caller_passwd* is the System Security Officer's password.

new_passwd – is the new password for the user, or for *loginame*. It must be at least 6 bytes long. Enclose passwords that include characters besides A-Z, a-z, or 0-9 in quotation marks. Also enclose passwords that begin with 0-9 in quotes.

loginame – the login name of the user whose account password is being changed by the System Security Officer.

Examples

1. **sp_password "3blindmice", "2mediumhot"**

   Changes your password from password from "3blindmice" to "2mediumhot." (Enclose the passwords in quotes because they begin with numerals.)

2. **sp_password "2tomato", sesame1, victoria**

   A System Security Officer whose password is "2tomato" has changed Victoria's password to "sesame1."

3. **sp_password null, "16tons"**

   Changes your password from NULL to "16tons." Notice that NULL is not enclosed in quotes. (NULL is not a permissible new password.)

4. **PRODUCTION...sp_password figaro, lilacs**

   Changes your password on the PRODUCTION server from "figaro" to "lilacs."

Comments

- Any user can change his or her password with **sp_password.**
- New passwords must be at least 6 characters long. They cannot be NULL.
- The encrypted text of *caller_passwd* must match the existing encrypted password of the caller. If it does not, **sp_password** returns an error message and fails. *master.dbo.syslogins* lists passwords in encrypted form.
- If a client program requires users to have the same password on remote servers as on the local server, users must change their passwords on all the remote servers before changing their local passwords. Execute **sp_password** as a remote procedure call on each remote server. See example 4.
- You can set the **systemwide password expiration** configuration parameter to establish a password expiration interval that forces all Adaptive Server login accounts to change passwords on a regular basis. For more information, see the *System Administration Guide.*

Permissions

Only a System Security Officer can execute **sp_password** to change another user's password. Any user can execute **sp_password** to change his or her own password.

Tables Used

master.dbo.syslogins, *sysobjects*

See Also

System procedures	**sp_addlogin**, **sp_adduser**

sp_placeobject

Function

Puts future space allocations for a table or index on a particular segment.

Syntax

sp_placeobject *segname*, *objname*

Parameters

segname – is the name of the segment on which to locate the table or index.

objname – is the name of the table or index for which to place subsequent space allocation on the segment *segname*. Specify index names in the form "*tablename.indexname*".

Examples

1. **sp_placeobject segment3, authors**

   This command places all subsequent space allocation for the table *authors* on the segment named "segment3".

2. **sp_placeobject indexes, 'employee.employee_nc'**

   This command places all subsequent space allocation for the *employee* table's index named *employee_nc* on the segment named *indexes*.

Comments

- You cannot change the location of future space allocations for system tables.
- Placing a table or an index on a particular segment does not affect the location of any existing table or index data. It affects only future space allocation. Changing the segment used by a table or an index can spread the data among multiple segments.
- If you use **sp_placeobject** with a clustered index, the table moves with the index.
- You can specify a segment when you create a table or an index with **create table** or **create index**. If you do not specify a segment, the data goes on the *default* segment.

- When **sp_placeobject** splits a table or an index across more than one disk fragment, the diagnostic command **dbcc** displays messages about the data that resides on the fragments that were in use for storage before **sp_placeobject** executed. Ignore those messages.
- You cannot use **sp_placeobject** on a partitioned table.

Permissions

Only the table owner, Database Owner, or System Administrator can execute **sp_placeobject**.

Tables Used

sysindexes, *sysobjects*, *syspartitions*, *syssegments*

See Also

Commands	**alter table**, **dbcc**
System procedures	**sp_addsegment**, **sp_dropsegment**, **sp_extendsegment**, **sp_help**, **sp_helpindex**, **sp_helpsegment**

sp_plan_dbccdb

Function

Recommends suitable sizes for new *dbccdb* and *dbccalt* databases, lists suitable devices for *dbccdb* and *dbccalt*, and suggests a cache size and a suitable number of worker processes for the target database.

Syntax

sp_plan_dbccdb [*dbname*]

Parameters

dbname – specifies the name of the target database. If *dbname* is not specified, **sp_plan_dbccdb** makes recommendations for all databases in *master..sysdatabases*.

Example

1. sp_plan_dbccdb master

```
Recommended size for dbccdb is 4MB.

dbccdb database already exists with size 8MB.

Recommended values for workspace size, cache size and process
count are:

dbname          scan ws    text ws    cache    process count


master          64K        64K       640K    1
```

Returns configuration recommendations for creating a *dbccdb* database suitable for checking the *master* database. The *dbccdb* database already existed at the time this command was run, so the size of the existing database is provided for comparison.

2. sp_plan_dbccdb

```
Recommended minimum size for dbccdb is 4MB.

Recommended values for workspace size, cache size and process
count are:

dbname          scan ws    text ws    cache    process count

master          64K        64K        640K     1
tempdb          64K        64K        640K     1
model           64K        64K        640K     1
sybsystemprocs  272K       80K        640K     1
dbccdb         128K       64K       640K     1
```

Returns configuration recommendations for creating a *dbccdb* database suitable for checking all databases in the server. No *dbccdb* database existed at the time this command was run.

3. sp_plan_dbccdb pubs2

```
Recommended size for dbccdb is 4MB.

Recommended devices for dbccdb are:

Logical Device Name      Device Size
       Physical Device Name

sprocdev                 28672
      /remote/sybase/devices/srv_sprocs_dat
tun_dat                  8192
      /remote/sybase/devices/srv_tun_dat
tun_log                  4096
      /remote/sybase/devices/srv_tun_log

Recommended values for workspace size, cache size and process
count are:

dbname      scan ws    text ws    cache    process count

pubs2      64K        64K        640K     1
```

Returns configuration recommendations for creating a *dbccdb* database suitable for checking *pubs2*.

Comments

- **sp_plan_dbccdb** recommends suitable sizes for creating new *dbccdb* and *dbccalt* databases, lists suitable devices for the new database,

and suggests cache size and a suitable number of worker processes for the target database.

- If you specify *dbccdb*, **sp_plan_dbccdb** recommends values for *dbccalt*, the alternate database. If you specify *dbccalt*, **sp_plan_dbccdb** recommends values for *dbccdb*.
- **sp_plan_dbccdb** does not report values for existing *dbccdb* and *dbccalt* databases. To gather configuration parameters for an existing *dbccdb* or *dbccalt* database, use **sp_dbcc_evaluatedb**.
- For information on the **dbcc** stored procedures for maintaining *dbccdb* and for generating reports from *dbccdb*, see Chapter 10, "dbcc Stored Procedures."

Permissions

Only the System Administrator or Database Owner can execute **sp_plan_dbccdb**. Only the System Administrator can execute **sp_plan_dbccdb** without specifying a database name.

Tables Used

master..sysdatabases, *master..sysdevices*, *master..sysusages*

See Also

Commands	**dbcc**
System procedures	**sp_dbcc_evaluatedb**

sp_poolconfig

Function

Creates, drops, resizes, and provides information about memory pools within data caches.

Syntax

To create a memory pool in an existing cache, or to change pool size:

```
sp_poolconfig cache_name [, "mem_size[P|K|M|G]",
   "config_poolK" [, "affected_poolK"]]
```

To change a pool's wash size:

```
sp_poolconfig cache_name, "io_size",
   "wash=size[P|K|M|G]"
```

To change a pool's asynchronous prefetch percentage:

```
sp_poolconfig cache_name, "io_size",
   "local async prefetch limit=percent"
```

Parameters

cache_name – is the name of an existing data cache.

mem_size – is the size of the memory pool to be created or the new total size for an existing pool, if a pool already exists with the specified I/O size. The minimum size of a pool is 512K. Specify size units with **P** for pages, **K** for kilobytes, **M** for megabytes, or **G** for gigabytes. The default is kilobytes.

config_pool – is the I/O size performed in the memory pool where the memory is to be allocated or removed.

Valid I/O sizes are **2K**, **4K**, **8K**, and **16K**.

affected_pool – is the size of I/O performed in the memory pool where the memory is to be deallocated. If *affected_pool* is not specified, the memory is taken from the 2K pool.

io_size – is the size of I/O performed in the memory pool where the wash size is to be reconfigured. The combination of cache name and I/O size uniquely identifies a memory pool.

wash=*size* – Changes the wash size (the point in the cache at which Adaptive Server writes dirty pages to disk) for a memory pool.

local async prefetch limit=*percent* – sets the percentage of buffers in the pool that can be used to hold buffers that have been read into cache by asynchronous prefetch, but that have not yet been used.

Examples

1. `sp_poolconfig pub_cache, "10M", "16K"`

   Creates a 16K pool in the data cache *pub_cache* with 10MB of space. All space is taken from the default 2K memory pool.

2. `sp_poolconfig pub_cache, "3M", "8K", "16K"`

   Moves 3MB of space to the 8K pool from the 16K pool of *pub_cache*.

3. `sp_poolconfig "pub_cache"`

   Reports the current configuration of *pub_cache*.

4. `sp_poolconfig pub_cache, "0K", "16K"`

   Removes the 16K memory pool from *pub_cache*, placing all of the memory assigned to it in the 2K pool.

5. `sp_poolconfig pub_cache, "2K", "wash=508K"`

   Changes the wash size of the 2K pool in *pubs_cache* to 508K.

6. ```
   sp_poolconfig pub_cache, "2K",
   "local async prefetch limit=15"
   ```

   Changes the asynchronous prefetch limit for the 2K pool to 15 percent.

Comments

- When you create a data cache with **sp_cacheconfig**, all space is allocated to the 2K memory pool. **sp_poolconfig** divides the data cache into additional pools with larger I/O sizes.
- If no large I/O memory pools exist in a cache, Adaptive Server performs I/O in 2K units, the size of a data page, for all of the objects bound to the cache. You can often enhance performance by configuring pools that perform large I/O. A 16K memory pool reads and writes eight data pages in a single I/O operation.
- The combination of cache name and I/O size must be unique. In other words, you can have only one pool of a given I/O size in a particular data cache.
- Only one **sp_poolconfig** command can be active on a single cache at one time. If a second **sp_poolconfig** command is issued before the first one completes, it sleeps until the first command completes.
   ```

- Figure 7-3 shows a data cache with:
  - The default data cache with a 2K pool and a 16K pool
  - A user cache with a 2K pool and a 16K pool
  - A log cache with a 2K pool and a 4K pool

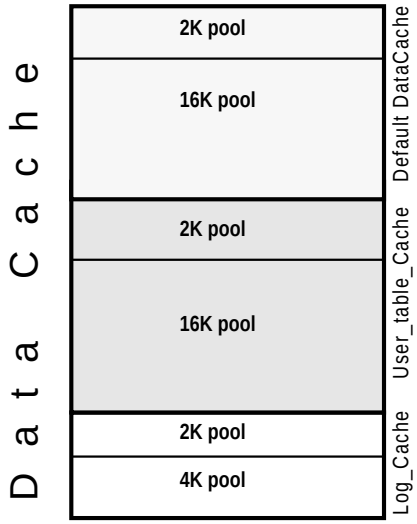

Figure 7-3: Data cache with default and user-defined caches

- You can create pools with I/O sizes up to 16K in the default data cache.
- The minimum size of a memory pool is 512K. You cannot reduce the size of any memory pool in any cache to less than 512K by transferring memory to another pool.
- Two circumstances can create pool less than 512K:
  - If you attempt to delete a pool by setting its size to zero, and some of the pages are in use, **sp_poolconfig** reduces the pool size as much as possible, and prints a warning message. The status for the pool is set to "Unavailable/deleted".
  - If you attempt to move buffers to create a new pool, and enough buffers cannot be moved to the new pool, **sp_poolconfig** moves as many buffers as it can, and the cache status is set to "Unavailable/too small."

In both of these cases, you can retry to command at a later time. The pool will also be deleted or be changed to the desired size when the server is restarted.

- You can create memory pools while Adaptive Server is active; no restart is needed for them to take effect. However, Adaptive Server can move only "free" buffers (buffers that are not in use or that do not contain changes that have not been written to disk). When you configure a pool or change its size, Adaptive Server moves as much memory as possible to the pool and prints an informational message showing the requested size and the actual size of the pool. After a restart of Adaptive Server, all pools are created at the configured size.
- The following commands perform only 2K I/O: **create database**, **alter database**, some **dbcc** commands, **disk init**, and **drop table. dbcc checktable** can perform large I/O, and **dbcc checkdb** performs large I/O on tables and 2K I/O on indexes. Also, recovery uses only the 2K memory pool: all pages are read into and changed in the 2K pool of the default cache. Be sure that your default 2K pool is large enough for these activities.
- Most Adaptive Servers perform best with 4K I/O configured for transactions logs. Adaptive Server uses the default I/O size of 4K if the default cache or a cache with a transaction log bound to it is configured with a 4K memory pool. Otherwise, it uses the 2K memory pool.
- You can increase the default log I/O size for a database using the **sp_logiosize** system procedure. However, the I/O size you specify must have memory pools of the same size in the cache bound to the transaction log. If not, Adaptive Server uses the 4K or 2K memory pools.

Wash Percentage

- The default value for the wash size is computed as follows:
  - If the pool size is less than 300MB, the default wash size is set to 20 percent of the buffers in the pool
  - If the pool size is greater than 300MB, the default wash size is 20 percent of the number of buffers in 300MB
- The minimum setting for the wash size is 10 buffers, and the maximum setting is 80 percent of the size of the pool.
- Each memory pool contains a wash area at the least recently used (LRU) end of the chain of buffers in that pool. Once dirty pages

(pages that have been changed while in cache) move into the wash area, Adaptive Server initiates asynchronous writes on these pages. The wash area must be large enough so that pages can be written to disk before they reach the LRU end of the pool. Performance suffers when Adaptive Server needs to wait for clean buffers.

The default percentage, placing 20 percent of the buffers in the wash area, is sufficient for most applications. If you are using an extremely large memory pool, and your applications have a very high data modification rate, you may want to increase the size to 1 or 2 percent of the pool. Contact Sybase Technical Support for more information about choosing an effective wash size.

Local Asynchronous Prefetch Percentage

- The default value for a pool's asynchronous prefetch percentage is set by the configuration parameter **global async prefetch limit**. The pool limit always overrides the global limit.
- To disable prefetch in a pool (if the global limit is a nonzero number), set the pool's limit to 0.
- For information on the performance impact of changes to the asynchronous prefetch limit, see the *Performance and Tuning Guide*.

Permissions

Only a System Administrator can execute **sp_poolconfig** to reconfigure memory pools within data caches. Any user can use **sp_poolconfig** to get information about memory pools.

Tables Used

master..sysconfigures

See Also

System procedures	**sp_cacheconfig**, **sp_helpcache**, **sp_unbindcache**, **sp_unbindcache_all**

sp_primarykey

Function

Defines a primary key on a table or view.

Syntax

```
sp_primarykey tabname, col1 [, col2, col3, ..., col8]
```

Parameters

tabname – is the name of the table or view on which to define the primary key.

col1 – is the name of the first column that makes up the primary key. The primary key can consist of from one to eight columns.

Examples

1. `sp_primarykey authors, au_id`

   Defines the *au_id* field as the primary key of the table *authors*.

2. `sp_primarykey employees, lastname, firstname`

   Defines the combination of the fields *lastname* and *firstname* as the primary key of the table *employees*.

Comments

- Executing **sp_primarykey** adds the key to the *syskeys* table. Only the owner of a table or view can define its primary key. **sp_primarykey** does not enforce referential integrity constraints; use the **primary key** clause of the **create table** or **alter table** command to enforce a primary key relationship.
- Define keys with **sp_primarykey**, **sp_commonkey**, and **sp_foreignkey** to make explicit a logical relationship that is implicit in your database design. An application program can use the information.
- A table or view can have only one primary key. To display a report on the keys that have been defined, execute **sp_helpkey**.
- The installation process runs **sp_primarykey** on the appropriate columns of the system tables.

Permissions

Only the owner of the specified table or view can execute **sp_primarykey.**

Tables Used

syscolumns, syskeys, sysobjects

See Also

Commands	**alter table**, **create table**, **create trigger**
System procedures	**sp_commonkey**, **sp_dropkey**, **sp_foreignkey**, **sp_helpjoins**, **sp_helpkey**

sp_processmail

(Windows NT only)

Function

Reads, processes, sends, and deletes messages in the Adaptive Server message inbox, using the **xp_findnextmsg**, **xp_readmail**, **xp_sendmail**, and **xp_deletemail** system extended stored procedures (ESPs).

Syntax

```
sp_processmail [subject] [, originator [, dbuser
   [, dbname [, filetype [, separator]]]]]
```

Parameters

subject – is the subject header of the message. If you specify a *subject* but not an *originator*, **sp_processmail** processes all unread messages in the inbox that has the specified subject header. If you specify both *subject* and *originator*, **sp_processmail** processes all unread messages with the specified subject header sent by the specified originator. If you do not specify either *subject* or *originator*, **sp_processmail** processes all the unread messages in the Adaptive Server message inbox.

originator – is the sender of an incoming message. If you specify an *originator* and do not specify a *subject*, **sp_processmail** processes all unread messages in the inbox sent by the specified originator.

dbuser – specifies the Adaptive Server login name to use for the user context for executing the query in the message. The default is "guest."

dbname – specifies the database name to use for the database context for executing the query in the message. The default is "master."

filetype – specifies the file extension of the attached file that contains the results of the query. The default is ".txt".

separator – specifies the character to use as a column separator in the query results. It is the same as the **/s** option of **isql**. The default is the tab character.

Examples

1. **sp_processmail @subject="SQL REPORT", @originator="janet", @dbuser="sa", @dbname="salesdb", @filetype="res", @separator=";"**

   Processes all unread messages in the Adaptive Server inbox with the subject header "SQL Report" submitted by mail user "janet", processes the received queries in the *salesdb* database as user "sa", and returns the query results to "janet" in a *.res* file attached to the mail message. The columns in the returned results are separated by semicolons.

2. **sp_processmail @dbuser="sa"**

   Processes all unread messages in the Adaptive Server inbox as user "sa" in the *master* database and returns the query results in *.txt* files, which are attached to the mail messages. The columns in the returned results are separated by tab characters.

Comments

- **sp_processmail** reads, processes, sends, and deletes messages in the Adaptive Server message inbox, using the **xp_findnextmsg**, **xp_readmail**, **xp_sendmail**, and **xp_deletemail** system ESPs.
- **sp_processmail** sends outgoing mail to the originator of the incoming mail message being processed.
- **sp_processmail** uses the default parameters when invoking the ESPs, except for the *dbuser*, *dbname*, *attachname*, and *separator* parameters to **xp_sendmail**, which can be overridden by the parameters to **sp_processmail**.
- **sp_processmail** processes all messages as Adaptive Server queries. It reads messages from the Adaptive Server inbox and returns query results to the sender of the message and all its cc'd and bcc'd recipients in an attachment to an Adaptive Server message. **sp_processmail** generates a name for the attached file consisting of "syb" followed by five random digits, followed by the extension specified by the *filetype* parameter; for example, "syb84840.txt."
- **sp_processmail** deletes messages from the inbox after processing them.
- The *subject* and *originator* parameters specify which messages should be processed. If neither of these parameters is supplied, **sp_processmail** processes all the unread messages in the Adaptive Server message inbox.

- **sp_processmail** does not process attachments to incoming mail. The query must be in the body of the incoming message.

Permissions

Only a System Administrator can execute **sp_processmail**.

See Also

System ESPs	**xp_deletemail, xp_findnextmsg, xp_readmail, xp_sendmail, xp_startmail**
Utility	**isql**

sp_procqmode

Function

Displays the query processing mode of a stored procedure, view, or trigger.

Syntax

sp_procqmode [*object_name* [, detail]]

Parameters

object_name – is the name of the stored procedure, view, or trigger whose query processing mode you are examining. If you do not specify an *object_name*, **sp_procqmode** reports on all procedures, views, and triggers in the current database.

detail – returns information about whether the object contains a subquery, and whether there is information about the object in *syscomments*.

Examples

1. sp_procqmode

```
Object Owner.name  Object Type        Processing Mode
-----------------  ----------------   --------------
dbo.au_info        stored procedure   pre-System 11
dbo.titleview      view               System 11 or later
```

Displays the query processing mode for all stored procedures in the current database.

2. sp_procqmode old_sproc, detail

```
Object Owner.Name     Object Type       Processing Mode     Subq Text
--------------------- ----------------- ------------------- ---- ----
dbo.au_info           stored procedure  pre-System 11       no   yes
```

Displays the query processing mode of the stored procedure *old_sproc*, reports whether *old_sproc* contains any subqueries, and reports whether *syscomments* has information about *old_sproc*.

3. sp_procqmode null, detail

Displays detailed reports for all objects in the database.

Comments

- The processing mode identifies whether the object was created in SQL Server release 10.0 or earlier. Objects created on release 10.x

(or earlier) servers are "pre-System 11" objects. Objects created on release 11.0 or later servers are "System 11 or later" objects.

- Subqueries in "pre-System 11" objects use a different processing mode than subqueries in "System 11 or later" objects. Upgrading to release 11.0 or later does not automatically change the processing mode of the subquery.

  In general, the "System 11 or later" processing mode is faster than "pre-System 11" processing mode. To change the processing mode to "System 11 or later", drop and re-create the object. You cannot create an object with "pre-System 11" processing on the current release of Adaptive Server, so you may want to create the object with another name and test it before dropping the version that uses "pre-System 11" processing mode.
- The processing mode displayed for a given object is independent of whether that object actually includes a subquery, and pertains only to the specified object, not to any dependent objects. You must check each object separately.
- The detailed report shows if the object contains a subquery, and reports if text is available in *syscomments* (for **sp_helptext** to report, or for the **defncopy** utility to copy out). **sp_procqmode** does not check that the text in *syscomments* is valid or complete.

Permissions

Only theDatabase Owner or object owner can execute **sp_procqmode**.

Tables Used

syscomments, *sysobjects*, *sysprocedures*

See Also

Stored Procedures	**sp_helptext**
Utility commands	**defncopy**

sp_procxmode

Function

Displays or changes the transaction modes associated with stored procedures.

Syntax

sp_procxmode [*procname* [, *tranmode*]]

Parameters

procname – is the name of the stored procedure whose transaction mode you are examining or changing.

tranmode – is the new transaction mode for the stored procedure. Values are **"chained"**, **"unchained"**, and **"anymode"**.

Examples

1. sp_procxmode

```
procedure name      user name   transaction mode
------------------  ---------   ----------------
byroyalty           dbo         Unchained
discount_proc       dbo         Unchained
history_proc        dbo         Unchained
insert_sales_proc   dbo         Unchained
insert_detail_proc  dbo         Unchained
storeid_proc        dbo         Unchained
storename_proc      dbo         Unchained
title_proc          dbo         Unchained
titleid_proc        dbo         Unchained
```

Displays the transaction mode for all stored procedures in the current database.

2. sp_procxmode byroyalty

```
procedure name                  transaction mode
------------------------------  ----------------
byroyalty                       Unchained
```

Displays the transaction mode of the stored procedure *byroyalty.*

3. sp_procxmode byroyalty, "chained"

Changes the transaction mode for the stored procedure *byroyalty* in the *pubs2* database from “unchained” to “chained”.

Comments

- To change the transaction mode of a stored procedure, you must be the owner of the stored procedure, the owner of the database containing the stored procedure, or the System Administrator. The Database Owner or System Administrator can change the mode of another user's stored procedure by qualifying it with the database and user name. For example:

  ```
  sp_procxmode "otherdb.otheruser.newproc", "chained"
  ```

- To use **sp_procxmode**, turn off chained transaction mode using the **chained** option of the **set** command. By default, this option is turned off.
- When you use **sp_procxmode** with no parameters, it reports the transaction modes of every stored procedure in the current database.
- To examine a stored procedure's transaction mode (without changing it), enter:

  ```
  sp_procxmode procname
  ```

- To change a stored procedure's transaction mode, enter:

  ```
  sp_procxmode procname, tranmode
  ```

- When you create a stored procedure, Adaptive Server tags it with the current session's transaction mode. This means:
  - You can execute "chained" stored procedures only in sessions using chained transaction mode.
  - You can execute "unchained" stored procedures only in sessions using unchained transaction mode.

  To execute a particular stored procedure in either chained or unchained sessions, set its transaction mode to "anymode".
- If you attempt to run a stored procedure under the wrong transaction mode, Adaptive Server returns a warning message, but the current transaction, if any, is not affected.

Permissions

Only a System Administrator, the Database Owner, or the owner of a procedure can execute **sp_procxmode** to change the transaction mode. Any user can execute **sp_procxmode** to display the transaction mode.

Tables Used

sysobjects

See Also

Commands	**begin transaction**, **commit**, **save transaction**, **set**

sp_recompile

Function

Causes each stored procedure and trigger that uses the named table to be recompiled the next time it runs.

Syntax

```
sp_recompile objname
```

Parameters

objname – is the name of a table in the current database.

Examples

```
1. sp_recompile titles
```

Recompiles each trigger and stored procedure that uses the table *titles* the next time the trigger or stored procedure is run.

Comments

- The queries used by stored procedures and triggers are optimized only once, when they are compiled. As you add indexes or make other changes to your database that affect its statistics, your compiled stored procedures and triggers may lose efficiency. By recompiling the stored procedures and triggers that act on a table, you can optimize the queries for maximum efficiency.
- **sp_recompile** looks for *objname* only in the current database and recompiles triggers and stored procedures only in the current database. **sp_recompile** does not affect objects in other databases that depend on the table.
- You cannot use **sp_recompile** on system tables.

Permissions

Any user can execute **sp_recompile**.

Tables Used

sysobjects

See Also

Commands	**create index**, **update statistics**

sp_remap

Function

Remaps a stored procedure, trigger, rule, default, or view from releases later than 4.8 and prior to 10.0 to be compatible with releases 10.0 and later. Use **sp_remap** on pre-existing objects that the upgrade procedure failed to remap.

Syntax

```
sp_remap objname
```

Parameters

objname – is the name of a stored procedure, trigger, rule, default, or view in the current database.

Examples

1. `sp_remap myproc`

   Remaps a stored procedure called *myproc*.

2. `sp_remap "my_db..default_date"`

   Remaps a rule called *default_date*. Execute a **use my_db** statement to open the *my_db* database before running this procedure.

Comments

- If **sp_remap** fails to remap an object, drop the object from the database and re-create it. Before running **sp_remap** on an object, it is a good idea to copy its definition into an operating system file with the **defncopy** utility. For more information about **defncopy**, see the *Utility Programs* manual for your platform.
- **sp_remap** can cause your transaction log to fill rapidly. Before running **sp_remap**, use the **dump transaction** command to dump the transaction log, as needed.
- You can use **sp_remap** only on objects in the current database.
- **sp_remap** makes no changes to objects that were successfully upgraded to the current release.

Permissions

Only a System Administrator or the owner of an object can execute **sp_remap**.

Tables Used

master.dbo.sysdatabases, sysobjects

See Also

Commands	**create default**, **create procedure**, **create rule**, **create trigger**, **create view**, **drop default**, **drop procedure**, **drop rule**, **drop trigger**, **drop view**, **dump transaction**
System procedures	**sp_helptext**
Utility programs	**defncopy**

sp_remoteoption

Function

Displays or changes remote login options.

Syntax

```
sp_remoteoption [remoteserver [, loginame
   [, remotename [, optname [, optvalue]]]]]
```

Parameters

remoteserver – is the name of the server that will be executing RPCs on this server.

➤ ***Note***

This manual page uses the term "local server" to refer to the server that is executing the remote procedures that are run from a "remote server".

loginame – is the login name that identifies the local login for the *remoteserver*, *loginame*, *remotename* combination.

remotename – is the remote user name that identifies the remote login for the *remoteserver*, *loginame*, *remotename* combination.

optname – is the name of the option to change. Currently, there is only one option, **trusted**, which means that the local server accepts remote logins from other servers without user-access verification for the particular remote login. The default is to use password verification. Adaptive Server understands any unique string that is part of the option name. Use quotes around the option name if it includes embedded blanks.

optvalue – is either **true** or **false. true** turns the option on, **false** turns it off.

Examples

1. sp_remoteoption

```
Settable remote login options.
remotelogin_option
------------------------
trusted
```

Displays a list of the remote login options.

2. **`sp_remoteoption GATEWAY, churchy, pogo, trusted, true`**

   Defines the remote login from the remote server GATEWAY to be **trusted** (that is, the password is not checked).

3. **`sp_remoteoption GATEWAY, churchy, pogo, trusted, false`**

   Defines the remote login "pogo" from the remote server GATEWAY as a login that is not trusted (that is, the password is checked).

4. **`sp_remoteoption GATEWAY, albert, NULL, trusted, true`**

   Defines all logins from GATEWAY that map to login "albert" on the local server to be trusted.

Comments

- To display a list of the remote login options, execute **sp_remoteoption** with no parameters.
- If you have used **sp_addremotelogin** to map all users from a remote server to the same local name, specify **trusted** for those users. For example, if all users from server GOODSRV that are mapped to "albert" are trusted, specify:

  ```
  sp_remoteoption GOODSRV, albert, NULL, trusted
    true
  ```

  If the logins are not specified as **trusted**, they cannot execute RPCs on the local server unless they specify local server passwords when they log into the remote server. When they use Open Client Client-Library, users can specify a password for server-to-server connections with the routine **ct_remote_pwd**. **isql** and **bcp** do not permit users to specify a password for RPC connections.

  If users are logged into the remote server using "unified login", the logins must also be trusted on the local server, or they must specify passwords for the server when they log into the remote server.

  For more information about setting up servers for remote procedure calls and for using "unified login", see the *System Administration Guide*.

Permissions

Only a System Security Officer can execute **sp_remoteoption**.

Tables Used

master.dbo.spt_values, *master.dbo.sysmessages*, *master.dbo.sysremotelogins*, *master.dbo.sysservers*, *sysobjects*

See Also

System procedures	**sp_addremotelogin**, **sp_dropremotelogin**, **sp_helpremotelogin**
Utility	**isql**

sp_remotesql

(Component Integration Services only)

Function

Establishes a connection to a remote server, passes a query buffer to the remote server from the client, and relays the results back to the client.

Syntax

```
sp_remotesql server, query
   [, query2, ... , query254]
```

Parameters

server_name – is the name of a remote server defined with **sp_addserver**.

query – is a query buffer a with maximum length of 255 characters.

query2 through *query254* – is a query buffer with a maximum length of 255 characters. If supplied, these arguments are concatenated with the contents of *query1* into a single query buffer.

Examples

1. ```
   sp_remotesql FREDS_SERVER, "select @@version"
   ```

   Passes the query buffer to FREDS_SERVER, which interprets **select @@version** and returns the result to the client. Adaptive Server does not interpret the result.

2. ```
   create procedure freds_version
   as
   exec sp_remotesql FREDS_SERVER, "select @@version"
   go
   exec freds_version
   go
   ```

   Illustrates the use of **sp_remotesql** in a stored procedure. This example and example 1 return the same information to the client.

```
3. sp_remotesql DCO_SERVER,
   "insert into remote_table
   (numbercol,intcol, floatcol,datecol )",
   "values (109.26,75, 100E5,'10-AUG-85')"
   select @@error
```

The server concatenates two query buffers into a single buffer, and passes the complete **insert** statement to the server DCO_SERVER for processing. The syntax for the **insert** statement is a format that DCO_SERVER understands. The returned information is not interpreted by the server. This example also examines the value returned in *@@error*.

```
4. declare @servname varchar(30)
   declare @querybuf varchar(200)
   select  @servname = "DCO_SERV"
   select  @querybuf = "select table_name
        from all_tables
        where owner = 'SYS'"
   exec sp_remotesql @servname, @querybuf
```

Illustrates the use of local variables as parameters to **sp_remotesql**.

Comments

- **sp_remotesql** establishes a connection to a remote server, passes a query buffer to the remote server from the client, and relays the results back to the client. The local server does not intercept results.
- You can use **sp_remotesql** within another stored procedure.
- The query buffer parameters must be a character expression with a maximum length of 255 characters. If you use a query buffer that is not *char* or *varchar*, you will receive datatype conversion errors.
- **sp_remotesql** sets the global variable *@@error* to the value of the last error message returned from the remote server if the severity of the message is greater than 10.
- If **sp_remotesql** is issued from within a transaction, Adaptive Server verifies that a transaction has been started on the remote server before passing the query buffer for execution. When the transaction terminates, the remote server is directed to commit the transaction. The work performed by the contents of the query buffer is part of the unit of work defined by the transaction.

  If transaction control statements are part of the query buffer, it is the responsibility of the client to ensure that the transaction

commit and **rollback** occur as expected. Mixing Transact-SQL with transaction control commands in the query buffer can cause unpredictable results.

- The local server manages the connection to the remote server. Embedding **connect to** or **disconnect** commands in the query buffer causes results that require interpretation by the remote server. This is not required or recommended. Typically, the result is a syntax error.

Permissions

Any user can execute **sp_remotesql**.

Tables Used

No tables are used.

See Also

Commands	**connect to...disconnect**
System procedures	**sp_autoconnect**, **sp_passthru**

sp_rename

Function

Changes the name of a user-created object or user-defined datatype in the current database.

Syntax

sp_rename *objname, newname*

Parameters

objname – is the original name of the user-created object (table, view, column, stored procedure, index, trigger, default, rule, check constraint, referential constraint, or user-defined datatype). If the object to be renamed is a column in a table, *objname* must be in the form "*table.column*". If the object is an index, *objname* must be in the form "*table.indexname*".

newname – is the new name of the object or datatype. The name must conform to the rules for identifiers and must be unique to the current database.

Examples

1. **sp_rename titles, books**

   Renames the *titles* table to *books*.

2. **sp_rename "books.title", bookname**

   Renames the *title* column in the *books* table to *bookname*.

3. **sp_rename "books.titleind", titleindex**

   Renames the *titleind* index in the *books* table to *titleindex*.

4. **sp_rename tid, bookid**

   Renames the user-defined datatype *tid* to *bookid*.

Comments

- **sp_rename** changes the name of a user-created object or datatype. You can change only the name of an object or datatype in the database in which you issue **sp_rename**.
- When you are renaming a column or index, do not specify the table name in the *newname*. See examples 2 and 3.

- You can change the name of a an object referenced by a view. For example, if a view references the *new_sales* table and you rename *new_sales* to *old_sales*, the view will reference *old_sales*.
- You cannot change the names of system objects and system datatypes.

◆ ***WARNING!***

Procedures, triggers, and views that depend on an object whose name has been changed work until they are dropped and re-created. Also, the old object name appears in query results until the user changes and re-creates the procedure, trigger, or view. Change the definitions of any dependent objects when you execute sp_rename. Find dependent objects with sp_depends.

Permissions

Only the Database Owner or a System Administrator can use the **setuser** command to assume another database user's identity to rename objects owned by other users. All users can execute **sp_rename** to rename their own objects.

Tables Used

syscolumns, *sysindexes*, *sysobjects*, *systypes*

See Also

Commands	**alter table**, **create default**, **create procedure**, **create rule**, **create table**, **create trigger**, **create view**
System procedures	**sp_addtype**, **sp_checkreswords**, **sp_depends**, **sp_renamedb**

sp_renamedb

Function

Changes the name of a user database.

Syntax

sp_renamedb *dbname*, *newname*

Parameters

dbname – is the original name of the database.

newname – is the new name of the database. Database names must conform to the rules for identifiers and must be unique.

Examples

1. **sp_renamedb accounting, financial**

   Renames the *accounting* database to *financial*.

2. ```
   sp_dboption work, single, true
   go
   use work
   go
   checkpoint
   go
   sp_renamedb work, workdb
   go
   use master
   go
   sp_dboption workdb, single, false
   go
   use workdb
   go
   checkpoint
   go
   ```

   Renames the database named *work*, which is a Transact-SQL reserved word, to *workdb*.

Comments

- **sp_renamedb** changes the name of a database. You **cannot** rename system databases or databases with external referential integrity constraints.
   ```

- The System Administrator must place a database in single-user mode with **sp_dboption** before renaming it and must restore it to multi-user mode afterward.
- **sp_renamedb** fails if any table in the database references, or is referenced by, a table in another database. Use the following query to determine which tables and external databases have foreign key constraints on primary key tables in the current database:

  ```
  select object_name(tableid), db_name(frgndbid)
  from sysreferences
  where frgndbid is not null
  ```

  Use the following query to determine which tables and external databases have primary key constraints for foreign key tables in the current database:

  ```
  select object_name(reftabid), db_name(pmrydbid)
  from sysreferences
  where pmrydbid is not null
  ```

  Use **alter table** to drop the cross-database constraints in these tables. Then, rerun **sp_renamedb**.
- When you change a database name:
  - Drop all stored procedures, triggers, and views that include the database name
  - Change the source text of the dropped objects to reflect the new database name
  - Re-create the dropped objects
  - Change all applications and SQL source scripts that reference the database, either in a **use** *database_name* command or as part of a fully qualified identifier (in the form *dbname*.[*owner*].*objectname*).
- If you use scripts to run **dbcc** commands or **dump database** and **dump transaction** commands on your databases, be sure to update those scripts.

◆ ***WARNING!***

Procedures, triggers, and views that depend on a database whose name has been changed work until they are re-created. Change the definitions of any dependent objects when you execute sp_renamedb. Find dependent objects with sp_depends.

Permissions

Only a System Administrator can execute **sp_renamedb**.

Tables Used

master.dbo.spt_values, *master.dbo.sysdatabases*, *sysobjects*

See Also

Commands	**create database**
System procedures	**sp_changedbowner**, **sp_dboption**, **sp_depends**, **sp_helpdb**, **sp_rename**

sp_rename_qpgroup

Function

Renames an abstract plan group.

Syntax

sp_rename_qpgroup ***old_name, new_name***

Parameters

old_name – is the current name of the abstract plan group.

new_name – is the new name for the group. The specified *new_name* cannot be the name of an existing abstract plan group in the database.

Examples

1. sp_rename_qpgroup dev_plans, prod_plans

Changes the name of the group from *dev_plans* to *prod_plans*.

Comments

- Use **sp_rename_qpgroup** to rename an abstract plan group. You cannot use the name of an existing plan group for the new name.
- **sp_rename_qpgroup** does not affect the contents of the renamed group. IDs of existing abstract plans are not changed.
- You cannot rename the default abstract plan groups, *ap_stdin* and *ap_stdout*.
- **sp_rename_qpgroup** cannot be run in a transaction.

Permissions

Only a System Administrator or the Database Owner can execute **sp_rename_qpgroup**.

Tables Used

sysattributes

See Also

System procedures	**sp_help_qpgroup**

sp_reportstats

Function

Reports statistics on system usage.

Syntax

sp_reportstats [*loginame*]

Parameters

loginame – is the login name of the user to show accounting totals for.

Examples

1. sp_reportstats

```
Name    Since         CPU     Percent CPU  I/O     Percent I/O
------  -----------   -----   ------------ -----   -------------
julie   jun 19 1993   10000   24.9962%     5000    24.325%
jason   jun 19 1993   10002   25.0013%     5321    25.8866%
ken     jun 19 1993   10001   24.9987%     5123    24.9234%
kathy   jun 19 1993   10003   25.0038%     5111    24.865%

      Total CPU   Total I/O
      ---------   ---------
      40006       20555
```

Displays a report of current accounting totals for all Adaptive Server users.

2. sp_reportstats kathy

```
Name    Since         CPU     Percent CPU   I/O     Percent I/O
------  -----------   -----   ------------  -----   -------------
kathy   Jul 24 1993   498     49.8998%      48392   9.1829%

      Total CPU   Total I/O
      ---------   ----------
      998         98392
```

Displays a report of current accounting totals for user “kathy.”

Comments

- **sp_reportstats** prints out the current accounting totals for all logins, as well as each login’s individual statistics and percentage of the overall statistics. **sp_reportstats** accepts one parameter, the login name of the account to report. With no parameters, **sp_reportstats** reports on all accounts.

- **sp_reportstats** does not report statistics for any process with a system user ID (*suid*) of 0 or 1. This includes deadlock detection, checkpoint, housekeeper, network, auditing, mirror handlers, and all users with **sa_role**.
- The units reported for "CPU" are **machine** clock ticks, not Adaptive Server clock ticks.
- The "probe" user exists for the two-phase commit probe process, which uses a challenge-and-response mechanism to access Adaptive Server.

Permissions

Only a System Administrator can execute **sp_reportstats**.

Tables Used

master.dbo.syslogins, *sysobjects*

See Also

System procedures	**sp_clearstats, sp_configure**

sp_revokelogin

(Windows NT only)

Function

Revokes Adaptive Server roles and default permissions from Windows NT users and groups when Integrated Security mode or Mixed mode (with Named Pipes) is active.

Syntax

```
sp_revokelogin {login_name | group_name}
```

Parameters

login_name – is the network login name of the Windows NT user.

group_name – is the Windows NT group name.

Examples

1. `sp_revokelogin jeanluc`

   Revokes all permissions from the Windows NT user named "jeanluc".

2. `sp_revokelogin Administrators`

   Revokes all roles from the Windows NT Administrators group.

Comments

- Use **sp_revokelogin** only when Adaptive Server is running in Integrated Security mode or Mixed mode, when the connection is Named Pipes. If Adaptive Server is running in Standard mode, or in Mixed mode using a connection other than Named Pipes, use the **revoke** command.
- If you revoke a user's roles and default privileges with **sp_revokelogin**, that user can no longer log into Adaptive Server over a trusted connection.

Permissions

Only a System Administrator can execute **sp_revokelogin**.

Tables Used

sysobjects

See Also

Commands	**grant**, **revoke**, **setuser**
System procedures	**sp_droplogin**, **sp_dropuser**, **sp_logininfo**

sp_role

Function

Grants or revokes roles to an Adaptive Server login account.

Syntax

```
sp_role {"grant" | "revoke"}, rolename, loginame
```

Parameters

grant | **revoke** – specifies whether to grant the role to or revoke the role from *loginame*.

rolename – is the role to be granted or revoked.

loginame – is the login account to or from which the role is to be granted or revoked.

Examples

1. `sp_role "grant", sa_role, alexander`

   Grants the System Administrator role to the login account named “alexander”.

Comments

- **sp_role** grants or revokes roles to an Adaptive Server login account.
- When you grant a role to a user, it takes effect the next time the user logs into Adaptive Server. Alternatively, the user can enable the role immediately by using the **set role** command. For example, the command:

  ```
  set role sa_role on
  ```

  enables the System Administrator role for the user.
- You cannot revoke a role from a user while the user is logged in.
- When users log in, all roles that have been granted to them are active (**on**). To turn a role off, use the **set** command. For example, to deactivate the System Administrator role, use the command:

  ```
  set role "sa_role" off
  ```

Permissions

Only a System Administrator can execute **sp_role** to grant the System Administrator role to other users. Only a System Security Officer can execute **sp_role** to grant any role other than "sa" to other users.

Tables Used

master.dbo.sysloginroles, master.dbo.syslogins, master.dbo.sysprocesses, master.dbo.syssrvroles, sysobjects

See Also

Commands	**grant**, **revoke**, **set**
Functions	**proc_role**
System procedures	**sp_displaylogin**

sp_sendmsg

Function

Sends a message to a UDP (User Datagram Protocol) port.

Syntax

sp_sendmsg *ip_address*, *port_number*, *message*

Parameters

ip_address – is the IP address of the machine where the UDP application is running.

port_number – is the port number of the UDP port.

Examples

```
1. sp_sendmsg "120.10.20.5", 3456, "Hello World"
```

Comments

- **sp_sendmsg** is not supported on Windows NT or NCR platforms.
- A System Security Officer must set the configuration parameter **allow sendmsg** to 1 to enable the use of UDP messaging.
- There are no security checks with **syb_sendmsg**. Sybase strongly recommends caution when using **syb_sendmsg** to send sensitive information across the network. By enabling this functionality, the user accepts any security problems which result from its use.
- This sample C program listens on a port that you specify and echoes the messages it receives. For example, to receive the **syb_sendmsg** calls for the example above, use:

  ```
  updmon 3456
  ```

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>

main(argc, argv)
int argc; char *argv[];
{
```

```
        struct sockaddr_in sadr;
        int portnum,sck,dummy,msglen;
        char msg[256];

        if (argc < 2) {
                printf("Usage: udpmon <udp portnum>\n");
                exit(1);
        }

        if ((portnum=atoi(argv[1])) < 1) {
                printf("Invalid udp portnum\n");
                exit(1);
        }

        if ((sck=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)) < 0) {
                printf("Couldn't create socket\n");
                exit(1);
        }

        sadr.sin_family = AF_INET;
        sadr.sin_addr.s_addr = inet_addr("0.0.0.0");
        sadr.sin_port = portnum;

        if (bind(sck,&sadr,sizeof(sadr)) < 0) {
                printf("Couldn't bind requested udp port\n");
                exit(1);
        }

        for (;;)
        {

if((msglen=recvfrom(sck,msg,sizeof(msg),0,NULL,&dummy)) < 0)
                        printf("Couldn't recvfrom() from udp
port\n");
                printf("%.*s\n", msglen, msg);
        }
}
```

Permissions

Any user can execute **sp_sendmsg**.

See Also

Functions	**syb_sendmsg**

sp_serveroption

Function

Displays or changes remote server options.

Syntax

```
sp_serveroption [server, optname, optvalue]
```

Parameters

server – is the name of the remote server for which to set the option.

optname – is the name of the option to be set or unset. Table 7-22 lists the option names.

Table 7-22: sp_serveroption options

Option	Meaning
net password encryption	Specifies whether to initiate connections with a remote server with the client side password encryption handshake or with the normal (unencrypted password) handshake sequence. The default is **false**, no network encryption.
readonly	Specifies that access to the server named is read only. This option is available only with Component Integration Services.
rpc security model A	The default model for handling RPCs. This model does not support mutual authentication, message integrity, or message confidentiality between the local server and the remote server.
timeouts	When unset (**false**), disables the normal timeout code used by the local server, so the site connection handler does not automatically drop the physical connection after one minute with no logical connection. The default is **true**.

Adaptive Server accepts any unique string that is part of the option name. Use quotes around the option name if it includes embedded blanks.

Examples

1. **sp_serveroption**

   ```
   Settable server options.

    ----------------------
   net password encryption
   readonly
   rpc security model A
   timeouts
   timeouts
   net password encryption
   ```

   Displays a list of the server options.

2. **sp_serveroption GATEWAY, "timeouts", false**

   Tells the server not to time out inactive physical connections with the remote server GATEWAY.

3. **sp_serveroption GATEWAY,**
   "net password encryption", true

   Specifies that when connecting to the remote server GATEWAY, GATEWAY sends back an encryption key to encrypt the password to send to it.

Comments

- To display a list of server options that can be set by the user, use **sp_serveroption** with no parameters.
- Once **timeouts** is set to **false**, the site handlers will continue to run until one of the two servers is shut down.
- The **net password encryption** option allows clients to specify whether to send passwords in plain text or encrypted form over the network when initiating a remote procedure call. If **net password encryption** is **true**, the initial login packet is sent without passwords, and the client indicates to the remote server that encryption is desired. The remote server sends back an encryption key, which the client uses to encrypt its passwords. The client then encrypts its passwords, and the remote server uses the key to authenticate them when they arrive.
- To set network password encryption for a particular **isql** session, you can use a command line option for **isql**. For more information, see the *Utility Programs* manual for your platform.
- You cannot use the **net password encryption** option when connecting to a pre-release 10.0 SQL Server.

- For more information on server options, see the *System Administration Guide.*

Permissions

Only a System Administrator can execute **sp_serveroption** to set the **timeouts** option. Any user can execute **sp_serveroption** with no parameters to display a list of options.

Tables Used

master.dbo.sysservers, *sysobjects*, *syssecmechs*

See Also

System procedures	**sp_helpserver**, **sp_password**
Utility	**isql**

sp_setlangalias

Function

Assigns or changes the alias for an alternate language.

Syntax

```
sp_setlangalias language, alias
```

Parameters

language – is the official language name of the alternate language.

alias – is the new local alias for the alternate language.

Examples

1. `sp_setlangalias french, français`

   This command assigns the alias name “français” for the official language name “french”.

Comments

- *alias* replaces the current value of *syslanguages.alias* for the official name.
- The **set language** command can use the new *alias* in place of the official language name.

Permissions

Only a System Administrator can execute **sp_setlangalias**.

Tables Used

master.dbo.syslanguages, *sysobjects*

See Also

Commands	set
System procedures	**sp_addlanguage**, **sp_droplanguage**, **sp_helplanguage**

sp_setpglockpromote

Function

Sets or changes the lock promotion thresholds for a database, for a table, or for Adaptive Server.

Syntax

```
sp_setpglockpromote {"database" | "table"}, objname,
   new_lwm, new_hwm, new_pct
sp_setpglockpromote server, NULL, new_lwm, new_hwm,
   new_pct
```

Parameters

server – sets server-wide values for the lock promotion thresholds.

"database" | **"table"** – specifies whether to set the lock promotion thresholds for a database or table. "database" and "table" are Transact-SQL keywords, so the quotes are required.

objname – is either the name of the table or database for which you are setting the lock promotion thresholds or **null**, if you are setting server-wide values.

new_lwm – specifies the value to set for the low watermark (LWM) threshold. The LWM must be less than or equal to the high watermark (HWM). The minimum value for LWM is 2. This parameter can be **null**.

new_hwm – specifies the value to set for the lock promotion HWM threshold. The HWM must be greater than or equal to the LWM. The maximum HWM is 2,147,483,647. This parameter can be **null**.

new_pct – specifies the value to set for the lock promotion percentage (PCT) threshold. PCT must be between 1 and 100. This parameter can be **null**.

Examples

1. `sp_setpglockpromote "server", NULL, 200, 300, 50`

Sets the server-wide lock promotion LWM to 200, the HWM to 300, and the PCT to 50.

2. **sp_setpglockpromote "database", master, 1000, 1100, 45**

   Sets lock promotion thresholds for the *master* database.

3. **sp_setpglockpromote "table", "pubs2..titles", 500, 700, 10**

   Sets lock promotion thresholds for the *titles* table in the *pubs2* database. This command must be issued from the *pubs2* database.

4. **sp_setpglockpromote "database", master, @new_hwm=1600**

   Changes the HWM threshold to 160 for the *master* database. The thresholds were previously set with **sp_setpglockpromote**. This command must be issued from the *master* database.

Comments

- **sp_setpglockpromote** configures the lock promotion values for a table, for a database, or for Adaptive Server.

  Adaptive Server acquires page locks on a table until the number of locks exceeds the lock promotion threshold. **sp_setpglockpromote** changes the lock promotion thresholds for an object, a database, or the server. If Adaptive Server is successful in acquiring a table lock, the page locks are released.

  When the number of locks on a table exceeds the HWM threshold, Adaptive Server attempts to escalate to a table lock. When the number of locks on a table is below the LWM, Adaptive Server does not attempt to escalate to a table lock. When the number of locks on a table is between the HWM and LWM and the number of locks exceeds the PCT threshold, Adaptive Server attempts to escalate to a table lock.

- Lock promotion thresholds for a table override the database or server-wide settings. Lock promotion thresholds for a database override the server-wide settings.

- Lock promotion thresholds for Adaptive Server do not need initialization, but you must initialize database and table lock promotion thresholds by specifying LWM, HWM, and PCT with **sp_setpglockpromote**, which creates a row for the object in *sysattributes* when it is first run for a database or table. Once the thresholds have been initialized, then they can be modified individually, as in example 4.

- For a table or a database, **sp_setpglockpromote** sets LWM, HWM, and PCT in a single transaction. If **sp_setpglockpromote** encounters an error while updating any of the values, then all changes are aborted and the transaction is rolled back. For server-wide changes, one or more thresholds may fail to be updated while others are successfully updated. Adaptive Server returns an error message if any values fail to be updated.
- To view the server-wide settings for the lock promotion thresholds, use **sp_configure "lock promotion"** to see all three threshold values. To view lock promotion settings for a database, use **sp_helpdb**. To view lock promotion settings for a table, use **sp_help**.

Permissions

Only a System Administrator can execute **sp_setpglockpromote**.

Tables Used

master.dbo.sysattributes, *master.dbo.sysconfigures*, *sysattributes*

See Also

System procedures	**sp_configure**, **sp_dropglockpromote**, **sp_help**, **sp_helpdb**

sp_setpsexe

Function

Sets custom execution attributes for a session while the session is active.

Syntax

```
sp_setpsexe spid, exeattr, value
```

Parameters

spid – is the ID of the session for which to set execution variables. Use **sp_who** to see *spid*s.

exeattr – identifies the execution attribute to be set. Values are **priority** and **enginegroup**.

value – is the new value of *exeattr*. Values for each attribute are as follows:

- If *exeattr* is **priority**, *value* is **HIGH**, **MEDIUM**, or **LOW**.
- If *exeattr* is **enginegroup**, *value* is the name of an existing engine group.

Examples

```
1. sp_setpsexe 1, "priority", "HIGH"
```

This statement sets the priority of the process with an ID of 1 to HIGH.

Comments

- Execution attribute values specified with **sp_setpsexe** are valid for the current session only and do not apply after the session terminates.
- Use **sp_setpsexe** with caution or it can result in degraded performance. Changing attributes "on the fly", using **sp_setpsexe**, can help if the process is not getting CPU time; however, if the performance problem is due to something else, such as locks, changing execution attributes could make the problem worse.
- Because you can only set execution attributes for sessions, **sp_setpsexe** cannot be set for a worker process *spid*.

- Except for the housekeeper *spid*, you cannot set execution attributes for system *spids*.
- **sp_setpsexe** does not work if there are no online engines in the associated engine group.

Permissions

Only a System Administrator can execute **sp_setpsexe** without restriction. Any user can execute **sp_setpsexe** to lower the priority of a process owned by that user.

Tables Used

sysattributes, sysprocesses

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe**

sp_set_qplan

Function

Changes the text of the abstract plan of an existing plan without changing the associated query.

Syntax

```
sp_set_qplan id, plan
```

Parameters

id – is the ID of the abstract plan.

plan – is a new abstract plan.

Examples

```
1. sp_set_qplan 563789159,
       "( g_join (scan t1) (scan t2))"
```

Comments

- Use **sp_set_qplan** to change the abstract plan of an existing plan. You can specify a maximum of 255 characters for a plan. If the abstract plan is longer than 255 characters, you can drop the old plan with **sp_drop_qplan** and then use **create plan** to create a new plan for the query.
- When you change a plan with **sp_set_qplan**, plans are not checked for valid abstract plan syntax. Also, the plan is not checked for compatibility with the SQL text. All plans modified with **sp_set_qplan** should be immediately checked for correctness by running the query for the specified ID.
- To find the ID of a plan, use **sp_help_qpgroup**, **sp_help_qplan**, or **sp_find_qplan**. Plan IDs are also returned by **create plan** and are included in **showplan** output.

Permissions

Any user can execute **sp_set_qplan** to change the text for a plan that he or she owns. Only the System Administrator or the Database Owner can change the text for a plan that belongs to another user.

Tables Used

sysqueryplans

See Also

Commands	**create plan**
System procedures	**sp_find_qplan**, **sp_help_qplan**

sp_setrowlockpromote

Function

Sets or changes row-lock promotion thresholds for a datarows-locked table, for all datarows-locked tables in a database, or for all datarows-locked tables on a server.

Syntax

```
sp_setrowlockpromote "server", NULL, new_lwm,
   new_hwm, new_pct
sp_setrowlockpromote {"database" | "table"}, objname,
   new_lwm, new_hwm, new_pct
```

Parameters

server – sets server-wide values for the row lock promotion thresholds.

"database" | **"table"** – specifies whether to set the row-lock promotion thresholds for a database or table.

objname – is either the name of the table or database for which you are setting the row-lock promotion thresholds or **null**, if you are setting server-wide values.

new_lwm – specifies the value to set for the low watermark (LWM) threshold. The LWM must be less than or equal to the high watermark (HWM). The minimum value for LWM is 2. This parameter can be **null**.

new_hwm – specifies the value to set for the high watermark (HWM) threshold. The HWM must be greater than or equal to the LWM. The maximum HWM is 2,147,483,647. This parameter can be **null**.

new_pct – specifies the value to set for the lock promotion percentage (PCT) threshold. PCT must be between 1 and 100. This parameter can be **null**.

Examples

1. `sp_setrowlockpromote "database", engdb, 400, 400,95`

   Sets row lock promotion values for all datarows-locked tables in the *engdb* database.

2. `sp_setrowlockpromote "table", sales, 250, 250, 100`

Sets row lock promotion values for the *sales* table.

Comments

- **sp_setrowlockpromote** sets or changes row-lock promotion thresholds for a table, a database, or Adaptive Server.

  Adaptive Server acquires row locks on a datarows-locked table until the number of locks exceeds the lock promotion threshold. If Adaptive Server is successful in acquiring a table lock, the row locks are released.

  When the number of row locks on a table exceeds the HWM, Adaptive Server attempts to escalate to a table lock. When the number of row locks on a table is below the LWM, Adaptive Server does not attempt to escalate to a table lock. When the number of row locks on a table is between the HWM and LWM, and the number of row locks exceeds the PCT threshold as a percentage of the number of rows in a table, Adaptive Server attempts to escalate to a table lock.
- Lock promotion is always two-tiered, that is, row locks are promoted to table locks. Adaptive Server does not promote from row locks to page locks.
- Lock promotion thresholds for a table override the database or server-wide settings. Lock promotion thresholds for a database override the server-wide settings.
- To change the lock promotion thresholds for a database, you must be using the *master* database. To change the lock promotion thresholds for a table in a database, you must be using the database where the table resides.
- Server-wide row lock promotion thresholds can also be set with **sp_configure**. When you use **sp_setrowlockpromote** to change the values server-wide, it changes the configuration parameters, and saves the configuration file. When you first install Adaptive Server, the server-wide row lock promotion thresholds set by the configuration parameters are:

row lock promotion HWM	200
row lock promotion LWM	200
row lock promotion PCT	100

  For more information, see the *System Administration Guide*.

- The system procedure **sp_sysmon** reports on row lock promotions.
- Database-level row lock promotion thresholds are stored in the *master..sysattributes* table. If you dump a database, and load it only another server, you must set the row lock promotion thresholds on the new server. Object-level row lock promotion thresholds are stored in the *sysattributes* table in the user database, and are included in the dump.

Permissions

Only a System Administrator can execute **sp_setrowlockpromote**.

Tables Used

master.dbo.sysattributes, master.dbo.sysconfigures, sysattributes

See Also

System procedures	**sp_droprowlockpromote**

sp_setsuspect_granularity

Function

Displays or sets the recovery fault isolation mode for a user database, which governs how recovery behaves when it detects data corruption.

Syntax

```
sp_setsuspect_granularity [dbname
   [, "database" | "page" [, "read_only"]]]
```

Parameters

dbname – is the name of the database for which to display or set the recovery fault isolation mode. For displaying, the default is the current database. For setting, you must be in the *master* database and specify the target *dbname*.

database – marks the entire database suspect, which makes it inaccessible, if the recovery process detects that any of its data is suspect.

page – marks only the corrupt pages suspect, making them inaccessible, if recovery detects corrupt data in the database. The rest of the data is accessible.

read_only –if specified, marks the entire database **read only** if recovery marks any pages suspect.

Examples

1. sp_setsuspect_granularity

```
DB Name  Cur. Suspect Gran.  Cfg. Suspect Gran.  Online mode
-------  -----------------  ------------------  -----------
pubs2    database           database            read/write
```

Displays the recovery fault isolation mode for the current database.

2. sp_setsuspect_granularity pubs2

Displays the current and configured recovery fault isolation mode for the *pubs2* database.

3. sp_setsuspect_granularity pubs2, "page"

```
DB Name       Cur. Suspect Gran. Cfg. Suspect Gran.
------------- ----------------- -----------------
pubs2          database             database
sp_setsuspect_granularity: The new values will become effective
      during the next recovery of the database 'pubs2'.
```

The next time recovery runs in the *pubs2* database, if any corrupt pages are detected, only the suspect pages will be taken offline and the rest of the database will be brought online.

4. sp_setsuspect_granularity pubs2, "page", "read_only"

The next time recovery runs in the *pubs2* database, if any corrupt pages are detected, only the suspect pages will be taken offline and the rest of the database will be brought online in read only mode.

5. sp_setsuspect_granularity pubs2, "database"

The next time recovery runs in the *pubs2* database, if any corrupt data is detected, the entire database will be marked suspect and taken offline.

Comments

- **sp_setsuspect_granularity** displays and sets the recovery fault isolation mode. This mode governs whether recovery marks an entire database or only the corrupt pages suspect when it detects that any data that it requires has been corrupted. For more information, see the *System Administration Guide*.
- The default recovery fault isolation mode of a user database is "database". You can set the recovery fault isolation mode only for a user database, not for a system database.
- You must be in the *master* database to set the recovery fault isolation mode.
- Data marked suspect due to corruption persists across Adaptive Server start-ups. When certain pages have been marked suspect, they remain offline after you reboot the server.
- When part or all of a database is marked suspect, the suspect data is not accessible to users unless a System Administrator has made the suspect data accessible with the **sp_forceonline_db** and **sp_forceonline_page** procedures.
- General database corruption, such as a corrupt database log or the unavailability of another resource not specific to a page,

causes the entire database to be marked suspect, even if the recovery fault isolation mode is "page".

- If you do not specify **page** or **database**, Adaptive Server displays the current and configured settings. The current setting is the one that was in effect the last time recovery was executed in the database. The configured setting is the one that will be in effect the next time recovery is executed in the database.
- If the database comes online in **read_only** mode, no user can modify any of its data, including data that is unaffected by the suspect pages and is thus online. However, the system administrator can make the database writeable using the **sp_dboption** system procedure to set **read only** to **false**. In this case, users could then modify the online data, but the suspect data would remain inaccessible.

Permissions

Only a System Administrator can execute **sp_setsuspect_granularity** to set the recovery fault isolation mode. Any user can execute **sp_setsuspect_granularity** to display the settings.

Tables Used

master.dbo.sysattributes, master.dbo.sysdatabases

See Also

Commands	**dump database, dump transaction, load database**
System procedures	**sp_forceonline_db, sp_forceonline_page, sp_listsuspect_db, sp_listsuspect_page, sp_setsuspect_threshold**

sp_setsuspect_threshold

Function

Displays or sets the maximum number of suspect pages that Adaptive Server allows in a database before marking the entire database suspect.

Syntax

```
sp_setsuspect_threshold [dbname [, threshold]]
```

Parameters

dbname – is the name of the database for which you want to display or set the suspect escalation threshold. The default is the current database.

threshold – indicates the maximum number of suspect data pages that recovery will allow before marking the entire database suspect. The default is 20 pages. The minimum is 0.

Examples

1. `sp_setsuspect_threshold pubs2, 5`

   Sets the maximum number of suspect pages to five. If there are more than five suspect pages, recovery will mark the entire database suspect.

2. `sp_setsuspect_threshold pubs2`

   Displays the current and configured settings for the suspect escalation threshold for the *pubs2* database.

3. `sp_setsuspect_threshold`

   Displays the current and configured settings for the recovery fault isolation threshold for the current user database.

Comments

- You must be in the *master* database to set the suspect escalation threshold with **sp_setsuspect_threshold**.
- If you do not specify the number of pages, Adaptive Server displays the current and configured settings. The current setting is the one that was in effect the last time recovery was executed in the database. The configured setting is the one that will be in effect the next time recovery is executed in the database.

Permissions

Only a System Administrator can execute **sp_setsuspect_threshold** to set the escalation threshold. Any user can execute **sp_setsuspect_threshold** to display the current settings.

Tables Used

master.dbo.sysattributes, master.dbo.sysdatabases

See Also

System procedures	**sp_forceonline_db, sp_forceonline_page, sp_listsuspect_db, sp_listsuspect_page, sp_setsuspect_granularity**

sp_showcontrolinfo

Function

Displays information about engine group assignments, bound client applications, logins, and stored procedures.

Syntax

sp_showcontrolinfo [*object_type, object_name, spid*]

Parameters

object_type – is **AP**, **LG**, **PR**, **EG**, or **PS** for application, login, stored procedure, engine group, or process, respectively. If you do not specify an *object_type* (or specify an *object_type* of **null**), **sp_showcontrolinfo** displays information about all types.

object_name – is the name of the application, login, stored procedure, or engine group. Do not specify an *object_name* if you specify **PS** as the *object_type*. If you do not specify an *object_name* (or specify an *object_name* of **null**), **sp_showcontrolinfo** displays information about all object names.

spid – is the Adaptive Server process ID. Specify an *spid* only if you specify **PS** as the *object_type*. If you do not specify an *spid* (or specify an *spid* of **null**), **sp_showcontrolinfo** displays information for all *spid*s. Use **sp_who** to see *spid*s.

Examples

1. sp_showcontrolinfo

Shows all user-assigned execution class-to-object bindings.

2. sp_showcontrolinfo 'AP', 'isql'

Displays the execution class of the **isql** application.

3. sp_showcontrolinfo 'PS'

Displays the execution class for all processes assigned to engine groups.

4. sp_showcontrolinfo 'PS', null, 7

Displays the execution class for *spid* 7.

Comments

- When used with no parameters, **sp_showcontrolinfo** displays information about all user-assigned engine group assignments, bound client applications, logins, and stored procedures. When used with the object_type parameter, **sp_showcontrolinfo** provides information on an individual basis about application, login, or stored procedure bindings to an execution class, engine group compositions, and session-level attribute bindings. For more information, see the *Performance and Tuning Guide*.
- Unless *object_type* is **PR**, execute **sp_showcontrolinfo** from the *master* database. If *object_type* is **PR**, execute **sp_showcontrolinfo** from the database in which the procedure resides.
- If *object_type* is **null**, **sp_showcontrolinfo** displays execution class information for objects that match the other parameters.
- If *object_name* is **null**, **sp_showcontrolinfo** displays the binding information for all applications, logins, and stored procedures.
- If *spid* is **null**, **sp_showcontrolinfo** displays execution class information for objects that match the other parameters.

Permissions

Any user can execute **sp_showcontrolinfo.**

Tables Used

sysattributes, syslogins

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_showexeclass, sp_showpsexe, sp_unbindexeclass**
Utility	**isql**

sp_showexeclass

Function

Displays the execution class attributes and the engines in any engine group associated with the specified execution class.

Syntax

sp_showexeclass [*execlassname*]

Parameters

execlassname – is the name of an execution class.

Examples

1. sp_showexeclass

```
classname   priority   engine_group   engines
---------  ---------- ------------- -----------
EC1         HIGH       ANYENGINE          ALL
EC2         MEDIUM     ANYENGINE          ALL
EC3         LOW        LASTONLINE         0
```

Displays the priority and engine group attribute values for all execution classes.

2. sp_showexeclass 'EC1'

```
classname   priority   engine_group           engines
----------- ---------- ---------------------- -----------
EC1         HIGH       ANYENGINE              ALL
```

Displays the attribute values of execution class *EC1*.

Comments

- **sp_showexeclass** displays the execution class attributes and the engines in any engine group associated with *execlassname*. For more information, see the *Performance and Tuning Guide.*
- If *execlassname* is NULL or absent, **sp_showexeclass** displays the priority and engine group attribute values for all execution classes, including the attribute values of the system-defined classes *EC1*, *EC2*, and *EC3*.

Permissions

Any user can execute **sp_showexeclass**.

Tables Used

sysattributes, sysengines

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_showcontrolinfo, sp_showpsexe, sp_unbindexeclass**

sp_showplan

Function

Displays the **showplan** output for any user connection for the current SQL statement or for a previous statement in the same batch.

Syntax

sp_showplan *spid*, *batch_id* output, *context_id* output, *stmt_num* output

To display the showplan output for the current SQL statement without specifying the *batch_id*, *context_id*, or *stmt_num*:

sp_showplan *spid*, null, null, null

Parameters

spid – is the process ID for any user connection. Use **sp_who** to see *spid*s.

batch_id – is a unique, nonnegative number for a batch

context_id – is a unique number for every procedure (or trigger) executed in a batch.

stmt_num – is the number of the current statement within a batch. The *stmt_num* must be a positive number.

Examples

1.
```
declare @batch int
declare @context int
declare @statement int
exec sp_showplan 99, @batch output, @context
output, @statement output
```

Displays the query plan for the current statement running in the user session with a *spid* value of 99, as well as values for the *batch_id*, *context_id*, and *statement_id* parameters. These values can be used to retrieve query plans in subsequent iterations of **sp_showplan** for the user session with a *spid* of 99.

2.
```
sp_showplan 99, null, null, null
```

Displays the **showplan** output for the current statement running in the user session with a *spid* value of 99.

Comments

- **sp_showplan** displays the **showplan** output for a currently executing SQL statement or for a previous statement in the same batch.
- To see the query plan for the previous statement within the same batch, execute **sp_showplan** again with the same parameter values, but subtract 1 from the statement number. Using this method, you can view all the statements in the statement batch back to query number one.
- **sp_showplan** can be run independently of Adaptive Server Monitor™ Server.
- If the *context_id* is greater than 0 for a SQL batch, the current statement is embedded in a stored procedure (or trigger) called from the original SQL batch. Select the *sysprocesses* row with the same *spid* value to display the procedure ID and statement ID.

Permissions

Only a System Administrator can execute **sp_showplan**.

Tables Used

None.

See Also

System procedures	**sp_who**

sp_showpsexe

Function

Displays execution class, current priority, and affinity for all client sessions running on Adaptive Server.

Syntax

sp_showpsexe [*spid*]

Parameters

spid – is the Adaptive Server session ID for which you want a report. The *spid* must belong to the application or login executing **sp_showpsexe**. Use **sp_who** to list *spid*s.

Examples

1. sp_showpsexe

```
spid  appl_name   login_name  exec_class current_priority task_affinity
----- ----------- ----------- ---------- ---------------- -------------
    1 isql        sa          EC1        HIGH             NONE
    5             NULL        NULL       LOW              NULL
    7 ctisql      sa          EC2        MEDIUM           NONE
    8 ctisql      sa          EC2        MEDIUM           NONE
```

Displays execution class, current priority, and affinity for all current client sessions.

2. sp_showpsexe 5

Displays the application name, login name, current priority, and engine affinity of the process with *spid* 5.

Comments

- **sp_showpsexe** displays execution class, current priority, and affinity for all sessions (objects with an *spid*). For more information, see the *Performance and Tuning Guide*.
- If the *spid* is NULL or absent, **sp_showpsexe** reports on all sessions currently running on Adaptive Server.
- **sp_showpsexe** does not report information for the following system processes: deadlock, checkpoint, network, auditing, and mirror handlers. It does display information for the housekeeper *spid*.

Permissions

Any user can execute **sp_showpsexe**.

Tables Used

syslogins, sysprocesses

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_showcontrolinfo, sp_showexeclass, sp_unbindexeclass**

sp_spaceused

Function

Displays estimates of the number of rows, the number of data pages, the size of indexes, and the space used by a specified table or by all tables in the current database.

Syntax

sp_spaceused [*objname* [,1]]

Parameters

objname – is the name of the table on which to report. If omitted, a summary of space used in the current database appears.

1 – prints separate information on the table's indexes and *text* / *image* storage.

Examples

1. sp_spaceused titles

```
name       rowtotal   reserved   data    index_size   unused
---------- ---------  ---------  ------- ----------   ----------
titles     18         46 KB      6 KB    4 KB         36 KB
```

Reports on the amount of space allocated (reserved) for the *titles* table, the amount used for data, the amount used for index(es), and the available (unused) space.

2. sp_spaceused titles, 1

```
index_name           size       reserved   unused
 -------------------- ---------- ---------- ----------
 titleidind           2 KB       32 KB      24 KB
 titleind             2 KB       16 KB      14 KB

 name       rowtotal   reserved   data   index_size unused
---------- --------  ---------  ------- ---------- ----------
titles           18      46 KB     6 KB       4 KB      36 KB
```

In addition to information on the *titles* table, prints information for each index on the table.

3. sp_spaceused blurbs,1

```
index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
blurbs               0 KB       14 KB      12 KB
tblurbs              14 KB      16 KB      2 KB

name       rowtotal reserved    data    index_size unused
---------- -------- ----------- ------- ---------- ----------
blurbs            6       30 KB    2 KB      14 KB      14 KB
```

Displays the space taken up by the *text/image* page storage separately from the space used by the table. The object name for *text/image* storage is "t" plus the table name.

4. sp_spaceused

```
database_name   database_size
--------------- --------------------------
master          5 MB

reserved    data        index_size  unused
---------   ---------   ----------- --------
2176 KB     1374 KB     72 KB       730 KB
```

Prints a summary of space used in the current database.

5. sp_spaceused syslogs

```
name       rowtotal   reserved   data    index_size   unused
---------- ---------  ---------  ------- ----------   ----------
syslogs    Not avail. 32 KB     32 KB   0 KB         0 KB
```

Reports on the amount of space reserved and the amount of space available for the transaction log.

Comments

- **sp_spaceused** displays estimates of the number of data pages, space used by a specified table or by all tables in the current database, and the number of rows in the tables. **sp_spaceused** computes the *rowtotal* value using the **rowcnt** built-in function. This function uses a value for the average number of rows per data page based on a value in the allocation pages for the object. This method is very fast, but the results are estimates, and update and insert activity change actual values. The **update statistics** command, **dbcc checktable**, and **dbcc checkdb** update the rows-per-page estimate, so *rowtotal* is most accurate after one of these commands executes. Always use **select count(*)** if you need exact row counts.

- **sp_spaceused** reports on the amount of space affected by tables, clustered indexes, and nonclustered indexes.
- The amount of space allocated (reserved) reported by **sp_spaceused** is a total of the data, index size, and available (unused) space.
- Space used by *text* and *image* columns, which are stored as separate database objects, is reported separately in the *index_size* column and is included in the summary line for a table. The object name for *text/image* storage in the *index_size* column is "t" plus the table name.
- When used on *syslogs*, **sp_spaceused** reports *rowtotal* as "Not available". See example 5.

Permissions

Any user can execute **sp_spaceused**.

Tables Used

master.dbo.spt_values, *master.dbo.sysusages*, *sysindexes*, *sysobjects*

See Also

Catalog stored procedures	**sp_statistics**
Commands	**create index**, **create table**, **drop index**, **drop table**
System procedures	**sp_help**, **sp_helpindex**

sp_syntax

Function

Displays the syntax of Transact-SQL statements, system procedures, utilities, and other routines for Adaptive Server, depending on which products and corresponding **sp_syntax** scripts exist on your server.

Syntax

sp_syntax *word* [, *mod*][, *language*]

Parameters

word – is the name or partial name of a command or routine; for example, "help", to list all system procedures providing help. To include spaces or Transact-SQL reserved words, enclose the word in quotes.

mod – is the name or partial name of one of the modules such as "Transact-SQL" or "Utility". Each **sp_syntax** installation script adds different modules. Use **sp_syntax** without any parameters to see which modules exist on your server.

language – is the language of the syntax description to be retrieved. *language* must be a valid language name in the *syslanguages* table.

Examples

1. sp_syntax

```
sp_syntax provides syntax help for Sybase products.
These modules are installed on this Server:

 Module
 --------------------
 OpenVMS
 Transact-SQL
 UNIX Utility
 System Procedure

Usage: sp_syntax command [, module [, language]]
```

Displays all **sp_syntax** modules available on your server.

2. sp_syntax "disk"

Displays the syntax and functional description of all routines containing the word or word fragment "disk". Since "disk" is a Transact-SQL reserved word, enclose it in quotes.

Comments

- The text for **sp_syntax** is in the database *sybsyntax*. Load **sp_syntax** and the *sybsyntax* database onto Adaptive Server with the installation script described in configuration documentation for your platform. If you cannot access **sp_syntax**, see your System Administrator for information about installing it on your server.
- You can use wildcard characters within the command name you are searching for. However, if you are looking for a command or function that contains the literal "_", you may get unexpected results, since the underscore wildcard character represents any single character.

Permissions

Any user can execute **sp_syntax**.

Tables Used

sybsyntax..sybsyntax

See Also

System procedures	**sp_help**, **sp_helpdb**

sp_sysmon

Function

Displays performance information.

Syntax

```
sp_sysmon begin_sample

sp_sysmon { end_sample | interval }
   [, section [, applmon] ]

sp_sysmon { end_sample | interval } [, applmon ]
```

Parameters

begin_sample – starts sampling. You cannot specify a section when you specify **begin_sample.**

end_sample – ends sampling and prints the report.

interval – specifies the time period for the sample. It must be in HH:MM:SS form, for example “00:20:00”.

section – is the abbreviation for one of the sections printed by **sp_sysmon**. Table 7-23 lists the values and corresponding names of the report sections.

Table 7-23: *sp_sysmon* report sections

Report Section	Parameter
Application Management	**appmgmt**
Data Cache Management	**dcache**
Disk I/O Management	**diskio**
ESP Management	**esp**
Index Management	**indexmgmt**
Kernel Utilization	**kernel**
Lock Management	**locks**
Memory Management	**memory**
Metadata Cache Management	**mdcache**
Monitor Access to Executing SQL	**monaccess**
Network I/O Management	**netio**

Table 7-23: *sp_sysmon* report sections (continued)

Report Section	Parameter
Parallel Query Management	**parallel**
Procedure Cache Management	**pcache**
Recovery Management	**recovery**
Task Management	**taskmgmt**
Transaction Management	**xactmgmt**
Transaction Profile	**xactsum**
Worker Process Management	**wpm**

applmon – specifies whether to print application detail, application and login detail, or no application detail. The default is to omit the application detail. Valid values are listed in Table 7-24.

Table 7-24: Values for *applmon* parameter to *sp_sysmon*

Parameter	Information Reported
appl_only	CPU, I/O, priority changes and resource limit violations by application name.
appl_and_login	CPU, I/O, priority changes and resource limit violations by application name and login name.
no_appl	Skips the by application or by login section of the report. This is the default.

This parameter is only valid when printing the full report and when you specify **appmgmt** for the *section*.

Examples

1. `sp_sysmon "00:10:00"`

Prints monitor information after 10 minutes.

2. `sp_sysmon "00:05:00", diskio`

Prints only the "Disk Management" section of the **sp_sysmon** report after 5 minutes.

3.
```
sp_sysmon begin_sample
go
execute proc1
go
execute proc2
go
select sum(total_sales) from titles
go
sp_sysmon end_sample, dcache
go
```

   Starts the sample, executes procedures and a query, ends the sample, and prints only the "Data Cache" section of the report.

4.
```
sp_sysmon "00:05:00", @applmon = appl_and_login
```

   Prints the full report and includes application and login detail for each login.

Comments

- **sp_sysmon** displays information about Adaptive Server performance. It sets internal counters to 0, then waits for the specified interval while activity on the server causes the counters to be incremented. When the interval ends, **sp_sysmon** prints information from the values in the counters. For more information, see the *Performance and Tuning Guide*.
- To print only a single section of the report, use the values listed in Table 7-24 for the second parameter.
- If you use **sp_sysmon** in batch mode, with **begin_sample** and **end_sample**, the time interval between executions must be at least one second. You can use **waitfor delay "00:00:01"** to lengthen the execution time of a batch.
- During the sample interval, results are stored in signed integer values. Especially on systems with many CPUs and high activity, these counters can overflow. If you see negative results in your **sp_sysmon** output, reduce your sample time.

Permissions

Only a System Administrator can execute **sp_sysmon**.

Tables Used

master.dbo.sysconfigures, master.dbo.syscurconfigs, master.dbo.sysdevices, master.dbo.ssysmonitors

sp_thresholdaction

Function

Executes automatically when the number of free pages on the log segment falls below the last-chance threshold, unless the threshold is associated with a different procedure. Sybase does not provide this procedure.

Syntax

When a threshold is crossed, Adaptive Server passes the following parameters to the threshold procedure by position:

```
sp_thresholdaction @dbname,
        @segment_name,
        @space_left,
        @status
```

Parameters

@dbname – is the name of a database where the threshold was reached.

@segment_name – is the name of the segment where the threshold was reached.

@space_left – is the threshold size, in 2K pages.

@status – is 1 for the last-chance threshold; 0 for all other thresholds.

Examples

```
1. create procedure sp_thresholdaction
        @dbname varchar(30),
        @segmentname varchar(30),
        @space_left int,
        @status int
   as
        dump transaction @dbname to tapedump1
```

Creates a threshold procedure for the last-chance threshold that dumps the transaction log to a tape device.

Comments

- **sp_thresholdaction** must be created by the Database Owner (in a user database), or a System Administrator (in the *sybsystemprocs* database), or a user with **create procedure** permission.

- You can add thresholds and create threshold procedures for any segment in a database.
- When the last-chance threshold is crossed, Adaptive Server searches for the **sp_thresholdaction** procedure in the database where the threshold event occurs. If it does not exist in that database, Adaptive Server searches for it in *sybsystemprocs*. If it does not exist in *sybsystemprocs*, it searches *master*. If Adaptive Server does not find the procedure, it sends an error message to the error log.
- **sp_thresholdaction** should contain a **dump transaction** command to truncate the transaction log.
- By design, the last-chance threshold allows enough free space to record a **dump transaction** command. There may not be enough space to record additional user transactions against the database. Only commands that are not recorded in the transaction log (**select**, fast **bcp**, **readtext**, and **writetext**) and commands that might be necessary to free additional log space (**dump transaction**, **dump database**, and **alter database**) can be executed. By default, other commands are suspended and a message is sent to the error log. To abort these commands rather than suspend them, use the **abort tran on log full** option of **sp_dboption** followed by the **checkpoint** command.

Waking Suspended Processes

- Once the **dump transaction** command frees sufficient log space, suspended processes automatically awaken and complete.
- If fast **bcp**, **writetext**, or **select into** have resulted in unlogged changes to the database since the last backup, the last-chance threshold procedure cannot execute a **dump transaction** command. When this occurs, use **dump database** to make a copy of the database, then use **dump transaction** to truncate the transaction log.
- If this does not free enough space to awaken the suspended processes, it may be necessary to increase the size of the transaction log. Use the **log on** option of the **alter database** command to allocate additional log space.
- As a last resort, System Administrators can use **sp_who** to determine which processes are suspended, then use the **kill** command to kill them.

See Also

Commands	**create procedure**, **dump transaction**
System procedures	**sp_addthreshold**, **sp_dboption**, **sp_dropthreshold**, **sp_helpsegment**, **sp_helpthreshold**, **sp_modifythreshold**

sp_transactions

Function

Reports information about active transactions.

Syntax

```
sp_tranactions ["xid", xid_value] |
   ["state", {"heuristic_commit" | "heuristic_abort"
   | "prepared" | "indoubt"} [, "xactname"]] |
   ["gtrid", gtrid_value]
```

Parameters

xid_value – is a transaction name from the *xactname* column of *master.dbo.systransactions*.

gtrid_value – is the global transaction ID name for a transaction coordinated by Adaptive Server.

Examples

1. sp_transactions

```
xactkey                         type        coordinator starttime
state            connection dbid   spid   loid
failover                   srvname                        namelen
xactname
------------------------------ ----------- -----------
--------------------
----------------- ---------- ------ ------ -----------
------------------------- ------------------------------ -------
------------------------------------------------------------
0x00000b1700040000dd6821390001 Local       None        Jun 1 1999 3:47PM
Begun             Attached        1   1      2
Resident Tx               NULL                                17
$user_transaction
0x00000b1700040000dd6821390001 Remote      ASTC        Jun 1 1999 3:47PM
Begun             NA              0   8      0
Resident Tx               caserv2                         108

00000b1700040000dd6821390001-aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002
```

Displays general information about all active transactions.

2. sp_transactions "xid", "00000b1700040000dd6821390001-aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-caserv1-0002"

```
xactkey                        type        coordinator starttime
state             connection dbid   spid   loid
failover                  srvname                        namelen
xactname
commit_node     parent_node
gtrid
------------------------------ ----------- -----------
--------------------
----------------- ---------- ------ ------ -----------
------------------------- ------------------------------ -------
------------------------------------------------------------------
-------------
-------------
-------------
0x00000b2500080000dd6821960001 External    ASTC        Jun 1 1999 3:47PM
Begun             Attached        1      8         139
Resident Tx               NULL                              108

00000b1700040000dd6821390001-aa01f04ebb9a-00000b1700040000dd6821390001-
aa01f04ebb9a-caserv1-caserv1-0002

caserv1         caserv1
00000b1700040000dd6821390001-aa01f04ebb9a
```

Displays detailed information for the specified transaction.

3. **sp_transactions "state", "prepared"**

   Displays general information about transactions that are in the "prepared" state.

4. **sp_transactions "state", "prepared", "xactname"**

   Displays only the transaction names of transactions that are in the "prepared" state.

5. **sp_transactions "gtrid", "00000b1700040000dd6821390001-aa01f04ebb9a"**

```
xactkey                        type        coordinator starttime
state             connection dbid   spid   loid
failover                  srvname                        namelen
xactname
commit_node
parent_node
------------------------------ ----------- -----------
--------------------
----------------- ---------- ------ ------ -----------
------------------------- ------------------------------ -------
```

```
------------------------------------------------------------------
-------------
-------------
0x00000b1700040000dd6821390001 Local       None        Jun 1 1999 3:47PM
Begun             Attached        1      1           2
Resident Tx               NULL                                17
$user_transaction

caserv1
caserv1
```

Displays status information for transactions having the specified global transaction ID.

Comments

- **sp_transactions** translates data from the *systransactions* table to display information about active transactions. *systransactions* itself comprises data in the **syscoordinations** table, as well as in-memory information about active transactions.
- **sp_transactions** with no keywords displays information about all active transactions.
- **sp_transactions** with the **xid** keyword displays the *gtrid*, *commit_node*, and *parent_node* columns only for the specified transaction.
- **sp_transactions** with the **state** keyword displays information only for the active transactions in the specified state.

  sp_transactions with both **xid** and **xactname** displays only the transaction names for transactions in the specified state.
- **sp_transactions** with the **gtrid** keyword displays information only for the transactions with the specified global transaction ID.
- **sp_transactions** replaces the **sp_xa_scan_xact** procedure provided with XA-Library and XA-Server products.
- For more information, see *Using Adaptive Server Distributed Transaction Management Features*.

Column Descriptions for *sp_transactions* Output

- The *xactkey* column shows the internal transaction key that Adaptive Server uses to uniquely identify the transaction.
- The *type* column indicates the type of transaction:
  - "Local" means that the transaction was explicitly started on the local Adaptive Server with a **begin transaction** statement.

  - "Remote" indicates a transaction executing on a remote Adaptive Server.
  - "External" means that the transaction has an external coordinator associated with it. For example, transactions coordinated by a remote Adaptive Server, MSDTC, or an X/Open XA transaction manager are flagged as "External."
  - "Dtx_State" is a special state for distributed transactions coordinated by Adaptive Server. It indicates that a transaction on the local server was either committed or aborted, but Adaptive Server has been unable to resolve a branch of that transaction on a remote participant. This may happen in cases where Adaptive Server loses contact with a server it is coordinating.

- The *coordinator* column indicates the method or protocol used to manage a distributed transaction:

sp_transactions "coordinator" value	Meaning
None	Transaction is not a distributed transaction and does not require a coordinating protocol.
ASTC	Transaction is coordinated using the Adaptive Server transaction coordination services.
XA	Transaction is coordinated by the X/Open XA-compliant transaction manager via the Adaptive Server XA-Library interface. Such transaction managers include Encina, CICS, and Tuxedo.
DTC	Transaction is coordinated by MSDTC.
SYB2PC	Transaction is coordinated using Sybase two-phase commit protocol.

- The *starttime* column indicates the time that the transaction started.
- The *state* column indicates the state of the transaction at the time **sp_transactions** ran:

sp_transactions "state" value	Meaning
Begun	Transaction has begun but no updates have been performed.
Done Command	Transaction completed an update command.

sp_transactions "state" value	Meaning
Done	X/Open XA transaction has finished modifying data.
Prepared	Transaction has successfully prepared.
In Command	Transaction is currently modifying data.
In Abort Cmd	Execution of the current command in the transaction has been aborted.
Committed	Transaction has successfully committed, and the commit log record has been written.
In Post Commit	Transaction has successfully committed, but is currently deallocating transaction resources.
In Abort Tran	The transaction is being aborted. This may happen either as a result of an explicit command, or because of a system failure.
In Abort Savept	The transaction is being rolled back to a savepoint.
Begun-Detached	Transaction has begun, but there is no thread currently attached to it.
Done Cmd-Detached	Transaction has finished modifying data, and no thread is currently attached to it.
Done-Detached	Transaction will modify no more data, and no thread is currently attached to it.
Prepared-Detached	Transaction has successfully prepared, and no thread is currently attached to it.
Heur Committed	Transaction has been heuristically committed using the **dbcc complete_xact** command.
Heur Rolledback	Transaction has been heuristically rolled back using the **dbcc complete_xact** command.

- The *connection* column indicates whether or not the transaction is currently associated with a thread:
  - "Attached" indicates that the transaction has an associated thread of control.
  - "Detached" indicates that there is no thread currently associated with the transaction. Some external transaction managers, such as CICS and TUXEDO, use the X/Open XA "suspend" and "join" semantics to associate different threads with the same transaction.

- The *dbid* column indicates the database ID of the database in which transaction started.
- The *spid* column indicates the server process ID associated with the transaction. If the transaction is "Detached," the "spid" value is 0.
- The *loid* column indicates the unique lock owner ID from *master.dbo.systransactions*.
- The *failover* column indicates the failover state for the transaction:
  - "Resident Tx" indicates that the transaction started and is executing on the same server. "Resident Tx" is displayed under normal operating conditions, and on systems that do not utilize Adaptive Server high availability features.
  - "Failed-over Tx" is displayed after there has been a failover to a secondary companion server. "Failed-over Tx" means that a transaction originally started on a primary server and reached the prepared state, but was automatically migrated to the secondary companion server (for example, as a result of a system failure on the primary server). The migration of a prepared transaction occurs transparently to an external coordinating service.
  - "Tx by Failover-Conn" indicates that there was an attempt to start the transaction on a designated server, but the transaction was instead started on the secondary companion server. This occurs when the original server has experienced a failover condition.
- The *srvname* column indicates the name of the remote server on which the transaction is executing. This column is only meaningful for remote transactions. For local and external transactions, *srvname* is null.
- The *namelen* column indicates the total length of the *xactname* value.
- *xactname* is the transaction name. For local transactions, the transaction name may be defined as part of the **begin transaction** command. External transaction managers supply unique transaction names in a variety of formats. For example, X/Open XA-compliant transaction managers supply a transaction ID (*xid*) consisting of a global transaction identifier and a branch qualifier, both of which are stored in *xactname*.
- For transactions coordinated by Adaptive Server, the *gtrid* column displays the global transaction ID. Transaction branches

that are part of the same distributed transaction share the same *gtrid*. You can use a specific *gtrid* with the **sp_transactions gtrid** keyword to determine the state of other transaction branches in the same distributed transaction.

sp_transactions cannot display the *gtrid* for transactions that have an external coordinator. For transactions coordinated by an X/Open XA-compliant transaction manager, MSDTC, or SYB2PC, the *gtrid* column shows the full transaction name supplied by the external coordinator.

- For transactions coordinated by Adaptive Server, the *commit_node* column indicates the server that executes the outermost block of the distributed transaction. This outermost block ultimately determines the commit status of all subordinate transactions.

  For transactions not coordinated by Adaptive Server, *commit_node* displays one of the values described in Table 7-25.

Table 7-25: Values for *commit_node* and *parent_node*

Value	Meaning
server_name	Commit or parent node is an Adaptive Server with the specified *server_name*.
XATM	Commit or parent node is an X/Open XA-compliant transaction manager.
MSDTCTM	Commit or parent node is MSDTC.
SYB2PCTM	Transaction is coordinated using SYB2PC protocol.

- For transactions coordinated by Adaptive Server, the *parent_node* column indicates the server that is coordinating the external transaction on the local server.

  For transactions not coordinated by Adaptive Server, *parent_node* displays one of the values described in Table 7-25.

➤ ***Note***

The values for *commit_node* and *parent_node* can be different, depending on the levels of hierarchy in the distributed transaction.

Permissions

Any user can execute **sp_transactions**.

Tables Used

spt_values, systransactions

See Also

System procedures	**sp_who**, **sp_lock**

sp_unbindcache

Function

Unbinds a database, table, index, *text* object, or *image* object from a data cache.

Syntax

```
sp_unbindcache dbname [,[owner.]tablename
   [, indexname | "text only"]]
```

Parameters

dbname – is the name of database to be unbound or the name of the database containing the objects to be unbound.

owner – is the name of the table's owner. If the table is owned by the Database Owner, the owner name is optional.

tablename – is the name of the table to be unbound from a cache or the name of a table whose index, *text* object, or *image* object is to be unbound from a cache.

indexname – is the name of an index to be unbound from a cache.

text only – unbinds *text* or *image* objects from a cache.

Examples

1. `sp_unbindcache pubs2, titles`

Unbinds the *titles* table from the cache to which it is bound.

2. `sp_unbindcache pubs2, titles, titleidind`

Unbinds the *titleidind* index from the from the cache to which it is bound.

3. `sp_unbindcache pubs2, au_pix, text`

Unbinds the *text* or *image* object for the *au_pix* table from the cache to which it is bound.

4. `sp_unbindcache pubs2, syslogs`

Unbinds the transaction log, *syslogs*, from its cache.

Comments

- When you unbind a database or database object from a cache, all subsequent I/O for the cache is performed in the default data

cache. All dirty pages in the cache being unbound are written to disk, and all clean pages are cleared from the cache. For more information, see the *Performance and Tuning Guide*.

- Cache unbindings take effect immediately and do not require a restart of the server.
- When you drop a database, table, or index, its cache bindings are automatically dropped.
- To unbind a database, you must be using the *master* database. For tables, indexes, *text* objects, or *image* objects, you must be using the database where the objects are stored.
- To unbind any system tables in a database, you must be using the database, and the database must be in single-user mode. Use the command:

  ```
  sp_dboption db_name, "single user", true
  ```

  See **sp_dpoption** for more information.
- The following procedures provide information about the bindings for their respective objects: **sp_helpdb** for databases, **sp_help** for tables, and **sp_helpindex** for indexes.
- **sp_helpcache** prints the names of objects bound to caches.
- **sp_unbindcache** needs to acquire an exclusive table lock when you are unbinding a table or its indexes to a cache. No pages can be read while the unbinding takes place. If a user holds locks on a table, and you issue **sp_unbindcache** on that object, the **sp_unbindcache** task sleeps until the locks are released.
- When you change the cache binding for an object with **sp_bindcache** or **sp_unbindcache**, the stored procedures that reference the object are recompiled the next time they are executed. When you change the binding for a database, the stored procedures that reference objects in the database are recompiled the next time they are executed.
- To unbind all objects from a cache, use the system procedure **sp_unbindcache_all**.

Permissions

Only a System Administrator can execute **sp_unbindcache**.

Tables Used

master..sysattributes, master..sysdatabases, sysindexes, sysobjects

See Also

System procedures	**sp_bindcache**, **sp_dboption**, **sp_helpcache**, **sp_unbindcache_all**

sp_unbindcache_all

Function

Unbinds all objects that are bound to a cache.

Syntax

sp_unbindcache_all *cache_name*

Parameters

cache_name – is the name of the data cache from which objects are to be unbound.

Examples

1. sp_unbindcache_all pub_cache

Unbinds all databases, tables, indexes, *text* objects and *image* objects that are bound to *pub_cache*.

Comments

- When you unbind entities from a cache, all subsequent I/O for the cache is performed in the default cache.
- To unbind individual objects from a cache, use the system procedure **sp_unbindcache**.
- See **sp_unbindcache** for more information about unbinding caches.

Permissions

Only a System Administrator can execute **sp_unbindcache_all**.

Tables Used

master..sysattributes, master..sysdatabases, sysindexes, sysobjects

See Also

System procedures	**sp_bindcache**, **sp_helpcache**, **sp_unbindcache**

sp_unbindefault

Function

Unbinds a created default value from a column or from a user-defined datatype.

Syntax

```
sp_unbindefault objname [, futureonly]
```

Parameters

objname – is the name of either the table and column or the user-defined datatype from which to unbind the default. If the parameter is not of the form *"table.column"*, then *objname* is assumed to be a user-defined datatype. When unbinding a default from a user-defined datatype, any columns of that type that have the same default as the user-defined datatype are also unbound. Columns of that type, whose default has already been changed, are unaffected.

futureonly – prevents existing columns of the specified user-defined datatype from losing their defaults. It is ignored when unbinding a default from a column.

Examples

1. `sp_unbindefault "employees.startdate"`

   Unbinds the default from the *startdate* column of the *employees* table.

2. `sp_unbindefault ssn`

   Unbinds the default from the user-defined datatype named *ssn* and all columns of that type.

3. `sp_unbindefault ssn, futureonly`

   Unbinds defaults from the user-defined datatype *ssn*, but does not affect existing columns of that type.

Comments

- Use **sp_unbindefault** to remove defaults created with **sp_bindefault.** Use **alter table** to drop defaults declared using the **create table** or **alter table** statements.

- Columns of a user-defined datatype lose their current default unless the default has been changed or the value of the optional second parameter is **futureonly.**
- To display the text of a default, execute **sp_helptext** with the default name as the parameter.

Permissions

Only the object owner can execute **sp_unbindefault.**

Tables Used

syscolumns, *sysobjects*, *sysprocedures*, *systypes*

See Also

Commands	**create default**, **drop default**
System procedures	**sp_bindefault**, **sp_helptext**

sp_unbindexeclass

Function

Removes the execution class attribute previously associated with an client application, login, or stored procedure for the specified scope.

Syntax

sp_unbindexeclass *object_name, object_type, scope*

Parameters

object_name – is the name of the application, login, or stored procedure for which to remove the association to the execution class.

object_type – identifies the type of *object_name* as **ap**, **lg**, or **pr** for application, login, or stored procedure.

scope – is the application name or the login name for which the unbinding applies for an application or login. It is the stored procedure owner name (user name) for stored procedures.

Examples

1. sp_unbindexeclass 'sa', 'lg', 'isql'

Removes the association between "sa" login scoped to application **isql** and an execution class. "sa" automatically binds itself to another execution class, depending on other binding specifications, precedence, and scoping rules. If no other binding is applicable, the object binds to the default execution class, *EC2*.

Comments

- The parameters must match an existing entry in the *sysattributes* system table.
- If you specify a null value for scope, Adaptive Server unbinds the object for which the scope is null, if there is one.
- A null value for scope does not indicate that unbinding should apply to all bound objects.
- When unbinding a stored procedure from an execution class, you must use the name of the stored procedure owner (user name) for the *scope* parameter.
- Stored procedures can be dropped before or after unbinding.

- A user cannot be dropped from a database if the user owns a stored procedure that is bound to an execution class in that database.
- Unbind objects of type **PR** before dropping them from the database.
- Unbinding will fail if the associated engine group has no online engines and active processes are bound to the associated execution class.
- Due to precedence and scoping rules, the execution class being unbound may or may not have been in effect for the object called *object_name*. The object automatically binds itself to another execution class, depending on other binding specifications and precedence and scoping rules. If no other binding is applicable, the object binds to the default execution class, *EC2*.

Permissions

Only a System Administrator can execute **sp_unbindexeclass.**

Tables Used

sysattributes, syslogins

See Also

System procedures	**sp_addengine, sp_addexeclass, sp_bindexeclass, sp_clearpsexe, sp_dropengine, sp_dropexeclass, sp_showcontrolinfo, sp_showexeclass, sp_showpsexe**
Utility	**isql**

sp_unbindmsg

Function

Unbinds a user-defined message from a constraint.

Syntax

sp_unbindmsg *constrname*

Parameters

constrname – is the name of the constraint from which a message is to be unbound.

Examples

1. sp_unbindmsg positive_balance

Unbinds a user-defined message from the constraint *positive_balance*.

Comments

- You can bind only one message to a constraint. To change the message bound to a constraint, use **sp_bindmsg**; the new message number replaces any existing bound message. It is not necessary to use **sp_unbindmsg** first.
- To retrieve message text from the *sysusermessages* table, execute **sp_getmessage**.

Permissions

Only the object owner can execute **sp_unbindmsg**.

Tables Used

sysconstraints, sysobjects

See Also

System procedures	**sp_addmessage**, **sp_bindmsg**, **sp_getmessage**

sp_unbindrule

Function

Unbinds a rule from a column or from a user-defined datatype.

Syntax

```
sp_unbindrule objname [, futureonly]
```

Parameters

objname – is the name of the table and column or of the user-defined datatype from which the rule is to be unbound. If the parameter is not of the form *"table.column"*, then *objname* is assumed to be a user-defined datatype. Unbinding a rule from a user-defined datatype also unbinds it from columns of the same type. Columns that are already bound to a different rule are unaffected.

futureonly – prevents columns of the specified user-defined datatype from losing their rules. It is ignored when unbinding a rule from a column.

Examples

1. ```
   sp_unbindrule "employees.startdate"
   ```

   Unbinds the rule from the *startdate* column of the *employees* table.

2. ```
   sp_unbindrule def_ssn
   ```

   Unbinds the rule from the user-defined datatype named *def_ssn* and all columns of that type.

3. ```
   sp_unbindrule ssn, futureonly
   ```

   The user-defined datatype *ssn* no longer has a rule, but existing *ssn* columns are unaffected.

Comments

- Executing **sp_unbindrule** removes a rule from a column or from a user-defined datatype in the current database. If you do not want to unbind the rule from existing *objname* columns, use **futureonly** as the second parameter.
- You cannot use **sp_unbindrule** to unbind a check constraint. Use **alter table** to drop the constraint.
- To unbind a rule from a table column, specify the *objname* argument in the form *"table.column"*.
   ```

- The rule is unbound from all existing columns of the user-defined datatype unless the rule has been changed or the value of the optional second parameter is **futureonly.**
- To display the text of a rule, execute **sp_helptext** with the rule name as the parameter.

Permissions

Only the object owner can execute **sp_unbindrule.**

Tables Used

syscolumns, *sysconstraints*, *sysobjects*, *sysprocedures*, *systypes*

See Also

Commands	**create rule**, **drop rule**
System procedures	**sp_bindrule**, **sp_helptext**

sp_volchanged

Function

Notifies the Backup Server that the operator performed the requested volume handling during a dump or load.

Syntax

```
sp_volchanged session_id, devname, action
   [, fname [, vname]]
```

Parameters

session_id – identifies the Backup Server session that requested the volume change. Use the *@session_id* parameter specified in the Backup Server's volume change request.

devname – is the device on which a new volume was mounted. Use the *@devname* parameter specified in the Backup Server's volume change request. If the Backup Server is not located on the same machine as the Adaptive Server, use the form:

```
device at backup_server_name
```

action – indicates whether the Backup Server should **abort**, **proceed** with, or **retry** the dump or load.

fname – is the file to be loaded. If you do not specify a file name with **sp_volchanged**, the Backup Server loads the **file** = *filename* parameter of the load command. If neither **sp_volchanged** nor the load command specifies which file to load, the Backup Server loads the first file on the tape.

vname – is the volume name that appears in the ANSI tape label. The Backup Server writes the volume name in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. If you do not specify a *vname* with **sp_volchanged**, the Backup Server uses the **dumpvolume** value specified in the dump command. If neither **sp_volchanged** nor the dump command specifies a volume name, the Backup Server leaves the name field of the ANSI tape label blank.

During loads, the Backup Server uses the *vname* to confirm that the correct tape has been mounted. If you do not specify a *vname* with **sp_volchanged**, the Backup Server uses the **dumpvolume** specified in the load command. If neither **sp_volchanged** nor the

load command specifies a volume name, the Backup Server does not check the name field of the ANSI tape label before loading the dump.

Examples

1. sp_volchanged 8, "/dev/nrmt4", RETRY

The following message from Backup Server indicates that a mounted tape's expiration date has not been reached:

```
Backup Server: 4.49.1.1: OPERATOR: Volume to be overwritten on
'/dev/rmt4' has not expired: creation date on this volume is
Sunday, Nov. 15, 1992, expiration date is Wednesday, Nov. 25,
1992.
Backup Server: 4.78.1.1: EXECUTE sp_volchanged
       @session_id = 8,
       @devname = '/auto/remote/pubs3/SERV/Masters/testdump',
       @action = { 'PROCEED' | 'RETRY' | 'ABORT' }
```

The operator changes the tape, then issues the command in example 1.

Comments

- If the Backup Server detects a problem with the currently mounted volume, it requests a volume change:
  - On OpenVMS systems, the Backup Server sends volume change messages to the operator terminal on the machine on which it is running. Use the **with notify = client** option of the dump or load command to route other Backup Server messages to the terminal session on which the **dump** or **load** request initiated.
  - On UNIX systems, the Backup Server sends messages to the client that initiated the dump or load request. Use the **with notify = operator_console** option of the dump or load command to route messages to the terminal where the Backup Server was started.
  - After mounting another volume, the operator executes **sp_volchanged** from any Adaptive Server that can communicate with the Backup Server performing the dump or load. The operator does not have to log into the Adaptive Server on which the dump or load originated.
- On OpenVMS systems, the operating system—not the Backup Server—requests a volume change when it detects the end of a volume or when the specified drive is offline. The operator uses the OpenVMS **REPLY** command to reply to these messages.

- On UNIX systems, the Backup Server requests a volume change when the tape capacity has been reached. The operator mounts another tape and executes **sp_volchanged**. Table 7-26 illustrates this process.

Table 7-26: Changing tape volumes on a UNIX system

Sequence	Operator, Using *isql*	Adaptive Server	Backup Server
1	Issues the **dump database** command		
2		Sends dump request to Backup Server	
3			Receives dump request message from Adaptive Server Sends message for tape mounting to operator Waits for operator's reply
4	Receives volume change request from Backup Server Mounts tapes Executes **sp_volchanged**		
5			Checks tapes If tapes are okay, begins dump When tape is full, sends volume change request to operator
6	Receives volume change request from Backup Server Mounts tapes Executes **sp_volchanged**		

Table 7-26: Changing tape volumes on a UNIX system (continued)

Sequence	Operator, Using *isql*	Adaptive Server	Backup Server
7			Continues dump When dump is complete, sends messages to operator and Adaptive Server
8	Receives message that dump is complete Removes and labels tapes	Receives message that dump is complete Releases locks Completes the **dump database** command	

Permissions

Any user can execute **sp_volchanged.**

Tables Used

master..sysdevices, sysobjects

See Also

Commands	**dump database**, **dump transaction**, **load database**, **load transaction**
Utility	**isql**

sp_who

Function

Reports information about all current Adaptive Server users and processes or about a particular user or process.

Syntax

sp_who [*loginame* | "*spid*"]

Parameters

loginame – is the Adaptive Server login name of the user you are requesting a report on.

spid – is the number of the process you are requesting a report on. Enclose process numbers in quotes (Adaptive Server expects a *char* type).

Examples

1. sp_who

```
fid spid  status    loginame origname hostname blk_spid  dbname
       cmd               blk_xloid
--- ----- --------   -------- -------- -------- -------  ------
       ------------------ ---------
  0    1  recv sleep  bird     bird     jazzy    0       master
       AWAITING COMMAND   0x0000ed92
  0    2  sleeping    NULL     NULL              0       master
       NETWORK HANDLER    0x0000ed92
  0    3  sleeping    NULL     NULL              0       master
       MIRROR HANDLER     0x0000ed92
  0    4  sleeping    NULL     NULL              0       master
       AUDIT PROCESS      0x0000ed92
  0    5  sleeping    NULL     NULL              0       master
       CHECKPOINT SLEEP   0x0000ed92
  0    6  recv sleep  rose     rose     petal    0       master
       AWAITING COMMAND   0x0000ed92
```

```
  0    7  sleeping    NULL     NULL     actor    0    sybsystemdb
       ASTC HANDLER       0x0000ed92
  0    8  running     robert   sa       helos    0       master
       SELECT             0x0000ed92
  0    9  send sleep  daisy    daisy    chain    0       pubs2
       SELECT             0x0000ed92
  0   10  alarm sleep lily     lily     pond     0       master
       WAITFOR            0x0000ed92
  0   11  lock sleep  viola    viola    cello    8       pubs2
       SELECT           0x0000ed92
```

Reports on the processes running on Adaptive Server. Process 11 (a **select** on a table) is blocked by process 8 (a **begin transaction** followed by an **insert** on the same table).

For process 8, the current *loginame* is "robert", but the original *loginame* is "sa". Login "sa" executed a **set proxy** command to impersonate the user "robert".

2. sp_who victoria

Reports on the processes being run by the user "victoria".

3. sp_who "17"

Reports what Adaptive Server process number 17 is doing.

4. sp_who

```
fid spid  status    loginame origname hostname blk_spid dbname
       cmd              block_xloid
---- ---- --------  -------- -------- -------- -------- ------
       ---------------- -----------
  0    1  running   sa       sa       helos    0        master
       SELECT             0
  0    2  sleeping  NULL     NULL              0        master
       NETWORK HANDLER    0
  0    3  sleeping  NULL     NULL              0        master
       DEADLOCK TUNE      0
  0    4  sleeping  NULL     NULL              0        master
       MIRROR HANDLER     0
  0    5  sleeping  NULL     NULL     actor    0        master
       ASTC HANDLER       0
  0    6  sleeping  NULL     NULL              0        master
       CHECKPOINT SLEEP   0
  0    5  sleeping  NULL     NULL              0        master
       HOUSEKEEPER       0
```

Reports on the processes running on Adaptive Server. Although no user processes other than **sp_who** are running, the server still shows activity. During idle cycles, the housekeeper task moves dirty buffers into the buffer wash region.

Comments

- **sp_who** reports information about a specified user or Adaptive Server process. Without parameters, **sp_who** reports which users are running what processes in all databases.
- If you enable mirrored disks or remote procedure calls, the mirror handler and the site handler also appear in the report from **sp_who**.
- The "spid" column contains the process identification numbers that are used in the Transact-SQL **kill** command. The "blk_spid" column contains the process IDs of the blocking process, if there is one. A blocking process (which may be infected or have an exclusive lock) is one that is holding resources needed by another process. The "block_xloid" column identifies the unique lock owner ID of a blocking transaction. The "fid" column identifies the family (including the coordinating process and its worker processes) to which a lock belongs (see **sp_familylock** for more information).
- Running **sp_who** on a single-engine server shows the **sp_who** process "running" and all other processes "runnable" or in one of the sleep states. In multi-engine servers, there can be a "running" process for each engine.
- **sp_who** reports NULL in the "loginame" column for all system processes.
- Evaluation of a conditional statement such as an **if** or **while** loop appears as "COND" in the "cmd" column.
- System Administrators can remove many processes with the **kill** command.

Permissions

Any user can execute **sp_who**.

Tables Used

master..sysprocesses

See Also

Commands	**kill**
System procedures	**sp_lock**

8 Catalog Stored Procedures

This chapter describes catalog stored procedures, which retrieve information from the system tables in tabular form.

Table 8-1 lists the catalog stored procedures that are covered in this chapter.

Table 8-1: Catalog stored procedures

Procedure	Description
sp_column_privileges	Returns permissions information for one or more columns in a table or view.
sp_columns	Returns information about the type of data that can be stored in one or more columns.
sp_databases	Returns a list of the databases in Adaptive Server.
sp_datatype_info	Returns information about a particular datatype or about all supported datatypes.
sp_fkeys	Returns information about foreign key constraints created in the current database with the **create table** or **alter table** command.
sp_pkeys	Returns information about primary key constraints created for a single table with the **create table** or **alter table** command.
sp_server_info	Returns a list of Adaptive Server attribute names and current values.
sp_special_columns	Returns the optimal set of columns that uniquely identify a row in a table or view; can also return a list of the columns that are automatically updated when any value in the row is updated by a transaction.
sp_sproc_columns	Returns information about a stored procedure's input and return parameters.
sp_statistics	Returns a list of indexes on a single table.
sp_stored_procedures	Returns information about one or more stored procedures.
sp_table_privileges	Returns privilege information for all columns in a table or view.
sp_tables	Returns a list of objects that can appear in a **from** clause.

Introduction to Catalog Stored Procedures

Catalog stored procedures retrieve information from the system tables in tabular form.

The catalog stored procedures, created by **installmaster** at installation, are located in the *sybsystemprocs* database and are owned by the System Administrator.

Many of them can be run from any database. If a catalog stored procedure is executed from a database other than *sybsystemprocs*, it retrieves information from the system tables in the database from which it was executed.

All catalog stored procedures execute at isolation level 1.

All catalog stored procedures report a return status. For example:

```
return status = 0
```

means that the procedure executed successfully. The examples in this book do not include the return status.

Specifying Optional Parameters

If a parameter value for a catalog stored procedure contains punctuation or embedded blanks, or is a reserved word, you must enclose it in single or double quotes. If the parameter is an object name qualified by a database name or owner name, enclose the entire name in single or double quotes.

➤ ***Note***

Do not use delimited identifiers as catalog stored procedure parameters. Doing so may produce unexpected results.

In many cases, it is more convenient to supply parameters to the catalog stored procedures in the form:

```
@parametername = value
```

than to supply all the parameters. The parameter names in the syntax statements match the parameter names defined by the procedures.

For example, the syntax for **sp_columns** is:

```
sp_columns table_name [, table_owner]
[, table_qualifier] [, column_name]
```

To use **sp_columns** to find information about a particular column, you can use:

```
sp_columns publishers, @column_name = "pub_id"
```

This provides the same information as the command with all of the parameters specified:

```
sp_columns publishers, "dbo", "pubs2", "pub_id"
```

You can also use "null" as a placeholder:

```
sp_columns publishers, null, null, "pub_id"
```

If you specify more parameters then the number of parameters expected by the system procedure, Adaptive Server ignores the extra parameters.

Pattern Matching

Adaptive Server offers a wide range of pattern matching through regular expressions. However, for maximum interoperability, assume only SQL standards pattern matching (the % and _ wildcard characters).

System Procedure Tables

These catalog stored procedures	Use these catalog stored procedure tables
sp_columns **sp_datatype_info** **sp_special_columns** **sp_sproc_columns**	*spt_datatype_info* *spt_datatype_info_ext* *spt_server_info*

in the *sybsystemprocs* database to convert internal system values such as status bits into human-readable format.

The catalog stored procedures **sp_column_privileges** and **sp_table_privileges** create and then drop temporary tables.

ODBC Datatypes

Table 8-2 and Table 8-3 list the datatype code numbers and matching datatype names returned by **sp_columns** and **sp_sproc_columns** in the "data_type" column. The source for the description is the Open

Database Connectivity (ODBC) Application Programming Interface (API).

Table 8-2: Code numbers for ODBC datatypes

Name	Type
char	1
decimal	3
double precision	8
float	6
integer	4
numeric	2
real	7
smallint	5
varchar	12

Table 8-3: Code numbers for extended datatypes

Name	Type
bigint	-5
binary (bit datatype)	-2
bit	-7
date	9
long varbinary	-4
long varchar	-1
time	10
timestamp	11
tinyint	-6
varbinary (bit-varying datatype)	-3

sp_column_privileges

Function

Returns permissions information for one or more columns in a table or view.

Syntax

```
sp_column_privileges table_name [, table_owner
   [, table_qualifier [, column_name]]]
```

Parameters

table_name – is the name of the table. The use of wildcard characters in pattern matching is not supported.

table_owner – is the name of the table owner. The use of wildcard characters in pattern matching is not supported. If you do not specify the table's owner, **sp_column_privileges** looks for a table owned by the current user and then for a table owned by the Database Owner.

table_qualifier – is the name of the database. Values are the name of the current database and **null**.

column_name – is the name of the column whose permissions you want to display. Use wildcard characters to request information for more than one column. If you do not specify a column name, permissions information for all columns in the specified table is returned.

Examples

1. sp_column_privileges discounts, null, null, discounttype

```
table_qualifier     table_owner    table_name          column_name
     grantor   grantee     privilege     is_grantable
------------------- ------------- ------------------- ----------------
     --------- ----------- ------------ ------------
pubs2               dbo            discounts           discounttype
     dbo       dbo          SELECT       YES
pubs2               dbo            discounts           discounttype
     dbo       dbo          UPDATE       YES
pubs2               dbo            discounts           discounttype
     dbo       dbo           REFERENCE   YES
pubs2               dbo            discounts           discounttype
     dbo       guest        SELECT       NO
pubs2               dbo            discounts           discounttype
     dbo       guest        UPDATE       NO
pubs2               dbo            discounts           discounttype
     dbo       guest        REFERENCE    NO
```

Comments

- Table 8-4 describes the results set:

Table 8-4: Results set for sp_column_privileges

Column	Datatype	Description
table_qualifier	*varchar(32)*	The name of the database in which the table specified for the *table_name* parameter is stored.
table_owner	*varchar(32)*	The table owner. If no value was specified for the *table_owner* parameter, this value is the current owner or the *dbo*.
table_name	*varchar(32)*	The name specified for the *table_name* parameter. This value cannot be NULL.
column_name	*varchar(32)*	The specified column name. If no column name was specified in the statement, the results include all columns in the specified table.
grantor	*varchar(32)*	The name of the database user who has granted permissions on *column_name* to *grantee*. This value cannot be NULL.
grantee	*varchar(32)*	The name of the database user who was granted permissions on *column_name* by *grantor*. This value cannot be NULL.

Table 8-4: Results set for sp_column_privileges (continued)

Column	Datatype	Description
privilege	*varchar(32)*	Identifies the column privilege. May be one of the following: SELECT - The grantee is permitted to retrieve data for the column. UPDATE - The grantee is permitted to update data in the column. REFERENCE - The grantee is permitted to refer to the column within a constraint (for example, a unique, referential, or table check constraint).
is_grantable	*varchar(3)*	Indicates whether the grantee is permitted to grant the privilege to other users. The values are YES, NO, and NULL.

Permissions

Any user can execute **sp_column_privileges**.

Tables Used

syscolumns, *sysobjects*, *sysusers*

See Also

System procedures	**sp_help**, **sp_helprotect**

sp_columns

Function

Returns information about the type of data that can be stored in one or more columns.

Syntax

sp_columns *table_name* [, *table_owner*]
[, *table_qualifier*] [, *column_name*]

Parameters

table_name – is the name of the table or view. Use wildcard characters to request information about more than one table.

table_owner – is the owner of the table or view. Use wildcard characters to request information about tables owned by more than one user. If you do not specify a table owner, **sp_columns** looks for tables owned by the current user and then for tables owned by the Database Owner.

table_qualifier – is the name of the database. This can be either the current database or NULL.

column_name – is the name of the column for which you want information. Use wildcard characters to request information about more than one column.

Examples

1. sp_columns "publishers", null, null, "p%"

```
table_qualifier               table_owner
     table_name                    column_name
     data_type type_name                    precision   length
     scale  radix  nullable
     remarks

     ss_data_type colid
----------------------------- -----------------------------
    ----------------------------- ----------------------------- --------
    ------------------------------ ----------- ----------- ------ ------
    --------
    ------------------------------------------------------------------------
------------------------------------------------------------------------
    ------------ -----
pubs2                         dbo
     publishers                    pub_id
```

```
          1 char                                    NULL          4
  NULL   NULL        0
NULL

           47     1
pubs2                          dbo
     publishers                     pub_name
          12 varchar                                NULL          40
  NULL   NULL        1
NULL

           39     2
```

Displays information about all columns in the *publishers* table that begin with "p".

2. `sp_columns "s%", null, null, "st%"`

Displays information about all columns beginning with "st" in tables that begin with "s".

Comments

- Table 8-5 shows the results set:

Table 8-5: Results set for sp_columns

Column	Datatype	Description
table_qualifier	*varchar(32)*	The name of the database in which the table specified for the *table_name* parameter is stored.
table_owner	*varchar(32)*	The table owner. If no value was specified for the *table_owner* parameter, this value is the current owner or the *dbo*.
table_name	*varchar(32)*	NOT NULL.
column_name	*varchar(32)*	NOT NULL.
data_type	*smallint*	Integer code for ODBC datatype. If this is a datatype that cannot be mapped into an ODBC type, it is NULL.
type_name	*varchar(30)*	String representing a datatype. The underlying DBMS presents this datatype name.
precision	*int*	Number of significant digits.
length	*int*	Length in bytes of a datatype.
scale	*smallint*	Number of digits to the right of the decimal point.

Table 8-5: Results set for sp_columns (continued)

Column	Datatype	Description
radix	*smallint*	Base for numeric datatypes.
nullable	*smallint*	The value 1 means NULL is possible; 0 means NOT NULL.
remarks	*varchar(254)*	
ss_data_type	*smallint*	An Adaptive Server datatype.
colid	*tinyint*	A column appended to the results set.
column_def	*varchar(255)*	
sql_data_type	*smallint*	
sql_datetime_sub	*smallint*	
char_octet_length	*int*	
ordinal_position	*int*	
is_nullable	*varchar(3)*	

- **sp_columns** reports the *type_name* as float, and *data_type* as 6 for columns defined as *double precision*. The Adaptive Server *double precision* datatype is a float implementation supports the range of values as specified in the ODBC specifications.

Permissions

Any user can execute **sp_columns.**

Tables Used

syscolumns, *sysobjects*, *systypes*, *sybsystemprocs..spt_datatype_info*

See Also

System procedures	**sp_help**

sp_databases

Function

Returns a list of databases in Adaptive Server.

Syntax

```
sp_databases
```

Parameters

None.

Examples

1. sp_databases

```
database_name       database_size     remarks
----------------    -------------     ------------
master                   5120         NULL
model                    2048         NULL
mydb                     2048         NULL
pubs2                    2048         NULL
sybsecurity              5120         NULL
sybsystemprocs          16384         NULL
tempdb                   2048         NULL
```

Comments

- Table 8-6 describes the results set:

Table 8-6: Results set for sp_databases

Column	Datatype	Description
database_name	*char*(32)	NOT NULL database name.
database_size	*int*	Size of database, in kilobytes.
remarks	*varchar*(254)	Adaptive Server always returns NULL.

Permissions

Any user can execute **sp_databases.**

Tables Used

master..sysdatabases, master..sysusages, sysobjects

See Also

System procedures	**sp_helpdb**

sp_datatype_info

Function

Returns information about a particular ODBC datatype or about all ODBC datatypes.

Syntax

```
sp_datatype_info [data_type]
```

Parameters

data_type – is the code number for the specified ODBC datatype about which information is returned. Datatype codes are listed in Table 8-2 and Table 8-3 on page 8-4.

Comments

- Table 8-7 describes the results set:

Table 8-7: Results set for sp_datatype_info

Column	Datatype	Description
type_name	*varchar(30)*	A DBMS-dependent datatype name (the same as the *type_name* column in the **sp_columns** results set).
data_type	*smallint*	A code for the ODBC type to which all columns of this type are mapped.
precision	*int*	The maximum precision for the datatype on the data source. Zero is returned for datatypes where precision is not applicable.
literal_prefix	*varchar(32)*	Character(s) used to prefix a literal. For example, a single quotation mark (') for character types and 0x for binary.
literal_suffix	*varchar(32)*	Character(s) used to terminate a literal. For example, a single quotation mark (') for character types and nothing for binary.
create_params	*varchar(32)*	A description of the creation parameters for this datatype.
nullable	*smallint*	The value 1 means this datatype can be created allowing null values; 0 means it cannot.

Table 8-7: Results set for sp_datatype_info (continued)

Column	Datatype	Description
case_sensitive	*smallint*	The value 1 means all columns of this type are case sensitive (for collations); 0 means they are not.
searchable	*smallint*	The value 1 means columns of this type can be used in a **where** clause.
unsigned_attribute	*smallint*	The value 1 means the datatype is unsigned; 0 means the datatype is signed.
money	*smallint*	The value 1 means it is a money datatype; 0 means it is not.
auto_increment	*smallint*	The value 1 means the datatype is automatically incremented; 0 means it is not.
local_type_name	*varchar(128)*	Localized version of the data source dependent name of the datatype.
sql_data_type	*smallint*	
sql_datetime_sub	*smallint*	
num_prec_radix	*smallint*	
interval_precision	*smallint*	

Permissions

Any user can execute **sp_datatype_info**.

Tables Used

sybsystemprocs..spt_datatype_info, *systypes*, *sysdatabases*, *sysmessages*, *sysprocesses*

See Also

System procedures	**sp_help**

sp_fkeys

Function

Returns information about foreign key constraints created with the **create table** or **alter table** command in the current database.

Syntax

```
sp_fkeys pktable_name [, pktable_owner]
   [, pktable_qualifier] [, fktable_name]
   [, fktable_owner] [, fktable_qualifier]
```

Parameters

pktable_name – is the name of the primary key table. The use of wildcard characters in pattern matching is not supported. You must specify either the *pktable_name* or the *fktable_name*, or both.

pktable_owner – is the name of the primary key table owner. The use of wildcard characters in pattern matching is not supported. If you do not specify the table owner, **sp_fkeys** looks for a table owned by the current user and then for a table owned by the Database Owner.

pktable_qualifier – is the name of the database that contains the primary key table. This can be either the current database or NULL.

fktable_name – is the name of the foreign key table. The use of wildcard characters in pattern matching is not supported. Either the *fktable_name* or the *pktable_name*, or both, must be given.

fktable_owner – is the name of the foreign key table owner. The use of wildcard characters in pattern matching is not supported. If an *fktable_owner* is not specified, **sp_fkeys** looks for a table owned by the current user and then for a table owned by the Database Owner.

fktable_qualifier – is the name of the database that contains the foreign key table. This can be either the current database or **null**.

Comments

- **sp_fkeys** returns information about foreign key constraints created with the **create table** or **alter table** command in the current database.

A foreign key is a key column in a table that logically depends on a **primary key** column in another table.

- Table 8-8 describes the results set:

Table 8-8: Results set for sp_fkeys

Column	Datatype	Description
pktable_qualifier	*varchar(32)*	The database that contains the primary key table.
pktable_owner	*varchar(32)*	The owner of the primary key table.
pktable_name	*varchar(32)*	NOT NULL.
pkcolumn_name	*varchar(32)*	NOT NULL.
fktable_qualifier	*varchar(32)*	The database that contains the foreign key table.
fktable_owner	*varchar(32)*	The owner of the foreign key table.
fktable_name	*varchar(32)*	NOT NULL.
fkcolumn_name	*varchar(32)*	NOT NULL.
key_seq	*smallint*	NOT NULL. The sequence number of the column in a multicolumn primary key.
update_rule	*smallint*	Action to be applied to the foreign key when the SQL operation is UPDATE. Zero is returned for this column.
delete_rule	*smallint*	Action to be applied to the foreign key when the SQL operation is DELETE. Zero is returned for this column.

- Both the primary key and foreign key must have been declared in a **create table** or **alter table** statement.
- If the primary key table name is supplied, but the foreign key table name is NULL, **sp_fkeys** returns all tables that include a foreign key to the given table. If the foreign key table name is supplied, but the primary key table name is NULL, **sp_fkeys** returns all tables that are related by a primary key / foreign key relationship to foreign keys in the foreign key table.
- **sp_fkeys** does not return information about keys declared with the **sp_commonkey**, **sp_foreignkey** or **sp_primarykey** system procedures.

Permissions

Any user can execute **sp_fkeys**.

Tables Used

sysobjects, *sysreferences*

See Also

Commands	**alter table**, **create table**
System procedures	**sp_helpkey**

sp_pkeys

Function

Returns information about primary key constraints created with the **create table** or **alter table** command for a single table.

Syntax

```
sp_pkeys table_name [, table_owner]
   [, table_qualifier]
```

Parameters

table_name – is the name of the table. The use of wildcard characters in pattern matching is not supported.

table_owner – is the name of the table owner. The use of wildcard characters in pattern matching is not supported. If *table_owner* is not specified, **sp_pkeys** looks for a table owned by the current user and then for a table owned by the Database Owner.

table_qualifier – is the name of the database that contains the table. This can be either the current database or NULL.

Comments

- Table 8-9 describes the results set:

Table 8-9: Results set for sp_pkeys

Column	Datatype	Description
table_qualifier	*varchar*(32)	The database name. This field can be NULL.
table_owner	*varchar*(32)	The table owner. If no value was specified for the *table_owner* parameter, this value is the current owner or the *dbo*.
table_name	*varchar*(32)	NOT NULL.
column_name	*varchar*(32)	NOT NULL.
key_seq	*smallint*	NOT NULL. The sequence number of the column in a multicolumn primary key.

- Primary keys must have been declared with the **create table** or **alter table** statement, not with the **sp_primarykey** system procedure.

- The term **primary key** refers to a logical primary key for a table. Adaptive Server expects that every logical primary key has a unique index defined on it and that this unique index is also returned in **sp_statistics**.

Permissions

Any user can execute **sp_pkeys.**

Tables Used

sysindexes, *sysobjects*

See Also

Commands	**alter table**, **create table**
System procedures	**sp_helpkey**

sp_server_info

Function

Returns a list of Adaptive Server attribute names and current values.

Syntax

sp_server_info [*attribute_id*]

Parameters

attribute_id – is the integer ID of the server attribute.

Examples

1. sp_server_info 12

```
attribute_id attribute_name           attribute_value
------------ ---------------------- -------------------------
          12 MAX_OWNER_NAME_LENGTH  0
```

2. sp_server_info

Returns the list of server attributes, described by the mandatory rows, and their values.

Comments

- Table 8-10 describes the results set:

Table 8-10: Results set for sp_server_info

Column	Datatype	Description
attribute_id	*int*	NOT NULL.
attribute_name	*varchar*(60)	NOT NULL.
attribute_value	*varchar*(255)	

- Table 8-11 shows the mandatory rows in the results set:

Table 8-11: Mandatory results returned by sp_server_info

ID	Server Attribute Name	Description	Value
1	DBMS_NAME	Name of the DBMS.	SQL SERVER
2	DBMS_VER	Version of the DBMS.	*@@version*

Table 8-11: Mandatory results returned by sp_server_info (continued)

ID	Server Attribute Name	Description	Value
6	DBE_NAME	Unused	
10	OWNER_TERM	Adaptive Server's term for a table owner (the second part of a three-part name).	`owner`
11	TABLE_TERM	Adaptive Server's term for a table (the third part of a three-part name).	`table`
12	MAX_OWNER_NAME_LENGTH	Maximum length of the name for a table owner (the second part of a three-part name).	`30`
16	IDENTIFIER_CASE	The case sensitivity of user-defined names (table names, column names, and stored procedure names) in the database (the case in which these objects are presented in the system catalogs).	`MIXED`
15	COLUMN_LENGTH	The maximum number of characters for a column name.	`30`
13	TABLE_LENGTH	The maximum number of characters for a table name.	`30`
100	USERID_LENGTH	The maximum number of characters for a user name.	`30`
17	TX_ISOLATION	The initial transaction isolation level the server assumes, corresponding to an isolation level defined in SQL92.	`4`
18	COLLATION_SEQ	The assumed ordering of the character set for this server.	
14	MAX_QUAL_LENGTH	Maximum length of the name for a table qualifier (the first part of a three-part table name).	`30`
101	QUALIFIER_TERM	Adaptive Server's term for a table qualifier (the first part of a three-part name).	`database`
19	SAVEPOINT_SUPPORT	Does the underlying DBMS support named savepoints?	`Y`
20	MULTI_RESULT_SETS	Does the underlying DBMS or the gateway itself support multiple results sets (can multiple statements be sent through the gateway, with multiple results sets returned to the client)?	`Y`
102	NAMED_TRANSACTIONS	Does the underlying DBMS support named transactions?	`Y`

Table 8-11: Mandatory results returned by sp_server_info (continued)

ID	Server Attribute Name	Description	Value
103	SPROC_AS_LANGUAGE	Can stored procedures be executed as language events?	Y
103	REMOTE_SPROC	Can stored procedures be executed through the remote stored procedure APIs in DB-Library?	Y
22	ACCESSIBLE_TABLES	In the **sp_tables** stored procedure, does the gateway return only tables, views, and so on, that are accessible by the current user (that is, the user who has at least **select** privileges for the table)?	Y
104	ACCESSIBLE_SPROC	In the **sp_stored_procedures** stored procedure, does the gateway return only stored procedures that are executable by the current user?	Y
105	MAX_INDEX_COLS	Maximum number of columns in an index for the DBMS.	32
106	RENAME_TABLE	Can tables be renamed?	Y
107	RENAME_COLUMN	Can columns be renamed?	Y
108	DROP_COLUMN	Can columns be dropped?	Y
109	INCREASE_COLUMN_LENGTH	Can column size be increased?	N
110	DDL_IN_TRANSACTION	Can DDL statements appear in transactions?	Y
111	DESCENDING_INDEXES	Are descending indexes supported?	Y
112	SP_RENAME	Can a stored procedure be renamed?	Y
500	SYS_SPROC_VERSION	The version of the catalog stored procedures currently implemented.	01.01.2822

Permissions

Any user can execute **sp_server_info**.

Tables Used

sybsystemprocs..spt_server_info, *sysobjects*

See Also

Catalog stored procedures	**sp_stored_procedures**, **sp_tables**

sp_special_columns

Function

Returns the optimal set of columns that uniquely identify a row in a table or view; can also return a list of *timestamp* columns, whose values are automatically generated when any value in the row is updated by a transaction.

Syntax

```
sp_special_columns table_name [, table_owner]
   [, table_qualifier] [, col_type]
```

Parameters

table_name – is the name of the table or view. The use of wildcard characters in pattern matching is not supported.

table_owner – is the name of the table or view owner. The use of wildcard characters in pattern matching is not supported. If you do not specify the table owner, **sp_special_columns** looks for a table owned by the current user and then for a table owned by the Database Owner.

table_qualifier – is the name of the database. This can be either the current database or NULL.

col_type – is **R** to return information about columns whose values uniquely identify any row in the table, or **V** to return information about *timestamp* columns, whose values are generated by Adaptive Server each time a row is inserted or updated.

Examples

1. sp_special_columns systypes

```
scope  column_name       data_type type_name     precision
         length      scale
 ------ --------------- --------- ------------- --------
        ----------- ------
      0 name                  12 varchar              30
              30   NULL
```

```
            2. sp_special_columns @table_name=authors, @col_type=R
scope  column_name         data_type type_name     precision
         length      scale
------ -------------------- --------- ------------ -----------
----------- ------
      0 au_id                      12 varchar                11
                 11   NULL
```

Comments

- Table 8-12 describes the results set:

Table 8-12: Results set for sp_special_columns

Column	Datatype	Description
scope	*int*	NOT NULL. Actual scope of the row ID. Adaptive Server always returns 0.
column_name	*varchar*(30)	NOT NULL. Column identifier.
data_type	*smallint*	The integer code for an ODBC datatype. If this datatype cannot be mapped to an ANSI/ISO type, the value is NULL. The native datatype name is returned in the *type_name* column. (See the ODBC datatypes Table 8-2 on page 8-4.)
type_name	*varchar*(13)	The string representation of the datatype. This is the datatype name as presented by the underlying DBMS.
precision	*int*	The number of significant digits.
length	*int*	The length in bytes of the datatype.
scale	*smallint*	The number of digits to the right of the decimal point.

Permissions

Any user can execute **sp_special_columns**.

Tables Used

sybsystemprocs..spt_datatype_info, *syscolumns*, *sysindexes*, *sysobjects*, *systypes*, *sysusers*

See Also

Datatypes	Timestamp Datatype
System procedures	**sp_help**

sp_sproc_columns

Function

Returns information about a stored procedure's input and return parameters.

Syntax

```
sp_sproc_columns procedure_name [, procedure_owner]
   [, procedure_qualifier] [, column_name]
```

Parameters

procedure_name – is the name of the stored procedure. The use of wildcard characters in pattern matching is not supported.

procedure_owner – is the owner of the stored procedure. The use of wildcard characters in pattern matching is not supported. If you do not specify the owner of the procedure, **sp_sproc_columns** looks for a procedure owned by the current user and then for a procedure owned by the Database Owner.

procedure_qualifier – is the name of the database. This can be either the current database or NULL.

column_name – is the name of the parameter about which you want information. If you do not supply a parameter name, **sp_sproc_columns** returns information about all input and return parameters for the stored procedure.

Comments

- Table 8-13 describes the results set:

Table 8-13: Results set for sp_sproc_columns

Column	Datatype	Description
procedure_qualifier	*varchar*(30)	
procedure_owner	*varchar*(30)	
procedure_name	*varchar*(41)	NOT NULL.
column_name	*varchar*(30)	NOT NULL.
column_type	*smallint*	

Table 8-13: Results set for sp_sproc_columns (continued)

Column	Datatype	Description
data_type	*smallint*	The integer code for an ODBC datatype. If this datatype cannot be mapped to an ANSI/ISO type, the value is NULL. The native datatype name is returned in the *type_name* column.
type_name	*char*(30)	The string representation of the datatype. This is the datatype name as presented by the underlying DBMS.
precision	*int*	The number of significant digits.
length	*int*	The length in bytes of the datatype.
scale	*smallint*	The number of digits to the right of the decimal point.
radix	*smallint*	Base for numeric types.
nullable	*smallint*	The value 1 means this datatype can be created allowing null values; 0 means it cannot.
remarks	*varchar*(254)	NULL.
ss_data_type	*tinyint*	An Adaptive Server datatype.
colid	*tinyint*	An Adaptive Server specific column appended to the result set.

- **sp_sproc_columns** reports the *type_name* as float, and *data_type* as 6 for parameters defined as *double precision*. The Adaptive Server *double precision* datatype is a float implementation supports the range of values as specified in the ODBC specifications.

Permissions

Any user can execute **sp_sproc_columns**.

Tables Used

sybsystemprocs..spt_datatype_info, *syscolumns*, *sysobjects*, *sysprocedures*, *systypes*

See Also

System procedures	**sp_help**, **sp_helptext**

sp_statistics

Function

Returns a list of indexes on a single table.

Syntax

```
sp_statistics table_name [, table_owner]
   [, table_qualifier] [, index_name] [, is_unique]
```

Parameters

table_name – is the name of the table. The use of wildcard character pattern matching is not supported.

table_owner – is the owner of the table. The use of wildcard character pattern matching is not supported. If *table_owner* is not specified, **sp_statistics** looks for a table owned by the current user and then for a table owned by the Database Owner.

table_qualifier – is the name of the database. This can be either the current database or NULL.

index_name – is the index name. The use of wildcard character pattern matching is not supported.

is_unique – is **Y** to return only unique indexes; otherwise, is **N** to return both unique and nonunique indexes.

Examples

1. sp_statistics publishers

```
table_qualifier                 table_owner
    table_name                      non_unique
    index_qualifier                 index_name
    type   seq_in_index column_name                     collation
    cardinality pages
------------------------------ ------------------------------
   ------------------------------ ----------
   ------------------------------ --------------------------
   ------ ------------ ------------------------------ --------
   ----------- -----------
```

```
pubs2                          dbo
    publishers                     NULL
    NULL                           NULL
        0         NULL NULL                          NULL
             3           1
pubs2                          dbo
    publishers                             0
    publishers                     pubind
         1            1 pub_id                        A
              3           1
```

Comments

- Table 8-14 describes the results set:

Table 8-14: Results set for sp_statistics

Column	Datatype	Description
table_qualifier	*varchar*(32)	The database name. This field can be NULL.
table_owner	*varchar*(32)	
table_name	*varchar*(32)	NOT NULL.
non_unique	*smallint*	NOT NULL. The value 0 means unique, and 1 means not unique.
index_qualifier	*varchar*(32)	
index_name	*varchar*(32)	
type	*smallint*	NOT NULL. The value 0 means clustered, 2 means hashed, and 3 means other.
seq_in_index	*smallint*	NOT NULL.
column_name	*varchar*(32)	NOT NULL.
collation	*char*(1)	The value A means ascending; D means descending; and NULL means not applicable.
cardinality	*int*	Number of rows in the table or unique values in the index.
pages	*int*	Number of pages to store the index or table.

- The indexes in the results set appear in ascending order, ordered by the *non-unique*, *type*, *index_name*, and *seq_in_index* columns.

- The index type *hashed* accepts exact match or range searches, but searches involving pattern matching do not use the index.

Permissions

Any user can execute **sp_statistics**.

Tables Used

syscolumns, *sysindexes*, *sysobjects*

See Also

System procedures	**sp_help**, **sp_helpindex**

sp_stored_procedures

Function

Returns information about one or more stored procedures.

Syntax

```
sp_stored_procedures [sp_name [, sp_owner
   [, sp_qualifier]]]
```

Parameters

sp_name – is the name of the stored procedure. Use wildcard characters to request information about more than one stored procedure.

sp_owner – is the owner of the stored procedure. Use wildcard characters to request information about procedures that are owned by more than one user.

sp_qualifier – is the name of the database. This can be the current database or NULL.

Comments

- **sp_stored_procedures** returns information about stored procedures in the current database only.
- Table 8-15 shows the results set:

Table 8-15: Results set for sp_stored_procedures

Column	Datatype	Description
procedure_qualifier	*varchar*(30)	The name of the database.
procedure_owner	*varchar*(30)	
procedure_name	*varchar*(41)	NOT NULL.
num_input_params	*int*	NOT NULL. Always returns -1.
num_output_params	*int*	NOT NULL. The value >= 0 shows the number of parameters; -1 means the number of parameters is indeterminate.
num_result_sets	*int*	NOT NULL. Always returns -1.
remarks	*varchar*(254)	NULL.

- **sp_stored_procedures** can return the name of stored procedures for which the current user does not have execute permission.owever, if the server attribute *accessible_sproc* is "Y" in the results set for **sp_server_info**, only stored procedures that are executable by the current user are returned.

Permissions

Any user can execute **sp_stored_procedures**.

Tables Used

sysobjects, *sysprocedures*, *sysprotects*, *sysusers*

See Also

System procedures	**sp_help**, **sp_helptext**

sp_table_privileges

Function

Returns privilege information for all columns in a table or view.

Syntax

```
sp_table_privileges table_name [, table_owner
   [, table_qualifier]]
```

Parameters

table_name – is the name of the table. The use of wildcard characters in pattern matching is not supported.

table_owner – is the name of the table owner. The use of wildcard characters in pattern matching is not supported. If you do not specify the table owner, **sp_table_privileges** looks for a table owned by the current user and then for a table owned by the Database Owner.

table_qualifier – is the name of the database. This can be either the current database or NULL.

Comments

- Table 8-16 shows the results set:

Table 8-16: Results set for sp_table_privileges

Column	Datatype	Description
table_qualifier	*varchar*(32)	The name of the database. This field can be NULL.
table_owner	*varchar*(32)	
table_name	*varchar*(32)	NOT NULL.
grantor	*varchar*(32)	NOT NULL.
grantee	*varchar*(32)	NOT NULL.

Table 8-16: Results set for sp_table_privileges (continued)

Column	Datatype	Description
privilege	*varchar*(32)	Identifies the table privilege. May be one of the following: SELECT - The grantee is permitted to retrieve data for one or more columns of the table. INSERT - The grantee is permitted to insert new rows containing data for one or more columns into the table. UPDATE - The grantee is permitted to update the data in one or more columns of the table. DELETE - The grantee is permitted to delete rows of data from the table. REFERENCE - The grantee is permitted to refer to one or more columns of the table within a constraint.
is_grantable	*varchar*(3)	Indicates whether the grantee is permitted to grant the privilege to other users. The values are YES, NO, and NULL.

Permissions

Any user can execute **sp_table_privileges**.

Tables Used

sysobjects, *sysusers*

See Also

System procedures	**sp_help**, **sp_helprotect**

sp_tables

Function

Returns a list of objects that can appear in a **from** clause.

Syntax

```
sp_tables [table_name] [, table_owner]
   [, table_qualifier][, table_type]
```

Parameters

table_name – is the name of the table. Use wildcard characters to request information about more than one table.

table_owner – is the table owner. Use wildcard characters to request information about more than one table.

table_qualifier – is the name of the database. Acceptable values are the name of the current database and NULL.

table_type – is a list of values, separated by commas, giving information about all tables of the table type(s) specified, including the following:

```
"'TABLE', 'SYSTEM TABLE', 'VIEW'"
```

➤ ***Note***

Enclose each table type with single quotation marks, and enclose the entire parameter with double quotation marks. Enter table types in uppercase.

Examples

```
1. sp_tables @table_type = "'TABLE', 'VIEW'"
```

This procedure returns information about all tables in the current database of the type TABLE and VIEW and excludes information about system tables.

Comments

- Adaptive Server does not necessarily check the read and write permissions on *table_name.* Access to the table is not guaranteed, even if you can display information about it.

- The results set includes tables, views, and synonyms and aliases for gateways to DBMS products.
- If the server attribute *accessible_tables* is "Y" in the results set for **sp_server_info**, only tables that are accessible by the current user are returned.
- Table 8-17 describes the results set:

Table 8-17: Results set for sp_tables

Column	Datatype	Description
table_qualifier	*varchar*(30)	The database name. This field can be NULL.
table_owner	*varchar*(30)	
table_name	*varchar*(30)	NOT NULL. The table name.
table_type	*varchar*(32)	NOT NULL. One of the following: 'TABLE', 'VIEW', 'SYSTEM TABLE'.
remarks	*varchar*(254)	NULL

Permissions

Any user can execute **sp_tables.**

Tables Used

sysdatabases, *sysobjects*, *sysprotects*, *sysusers*

See Also

System procedures	**sp_help**
Catalog stored procedures	**sp_server_info**

9 System Extended Stored Procedures

This chapter describes the system extended stored procedures (ESPs), which are extended stored procedures supplied by Sybase.

Table 9-1 lists the system extended stored procedures discussed in this chapter.

Table 9-1: System extended stored procedures

Procedure	Description	Platform
xp_cmdshell	Executes a native operating system command on the host system running Adaptive Server.	All Supporting DLLs
xp_deletemail	Deletes a message from the Adaptive Server message inbox.	NT Only
xp_enumgroups	Displays groups for a specific Windows NT domain.	NT Only
xp_findnextmsg	Retrieves the message identifier of the next message in the Adaptive Server message inbox.	NT Only
xp_logevent	Provides for logging a user-defined event in the Windows NT Event Log.	NT Only
xp_readmail	Reads a message from the Adaptive Server message inbox.	NT Only
xp_sendmail	Sends a message to the specified recipients using the MAPI interface.	NT Only
xp_startmail	Starts an Adaptive Server mail session.	NT Only
xp_stopmail	Stops an Adaptive Server mail session.	NT Only

Introduction

The system extended stored procedures, created by **installmaster** at installation, are located in the *sybsystemprocs* database and are owned by the System Administrator. They can be run from any database.

Permissions on System ESPs

Since system extended stored procedures are located in the *sybsystemprocs* database, their permissions are also set there.

Users with the **sa_role** have default execution permissions on the system ESPs. These System Administrators can grant execution permissions to other users.

DLLs associated with System ESPs

You can get the names of the DLLs associated with the system ESPs by running **sp_helpextendedproc** in the *sybsystemprocs* database.

Using System ESPs

The system ESPs follow the same calling conventions as the regular system procedures.

The only additional requirement for system ESPs is that the Open Server application, XP Server, must be running. Adaptive Server starts XP Server the first time an ESP is invoked. XP Server continues to run until you shut down Adaptive Server.

xp_cmdshell

Function

Executes a native operating system command on the host system running Adaptive Server.

Syntax

```
xp_cmdshell command [, no_output]
```

Parameters

command – is the operating system command string; maximum length is 255 bytes.

no_output – if specified, suppresses any output from the command.

Examples

1. `xp_cmdshell 'copy C:\log A:\log.0102', no_output`

   Silently copies the file named *log* on the C drive to a file named *log.0102* on the A drive.

2. `xp_cmdshell 'date'`

   Executes the operating system's **date** command and returns the current date as a row of data.

Comments

- **xp_cmdshell** returns any output, including operating system errors, as rows of text in a single column.
- **xp_cmdshell** is run from the current directory of the XP Server.
- The width of the column of returned output is 80 characters. The output is not formatted.
- **xp_cmdshell** cannot perform commands that require interaction with the user, such as "login".
- The user context in which an operating system command is executed via **xp_cmdshell** is controlled by the value of the **xp_cmdshell context** configuration parameter. If this parameter is set to 1 (the default), **xp_cmdshell** restricts permission to users with System Administration privileges at the operating system level. If this parameter is set to 0, **xp_cmdshell** uses the security context of the operating system account under which Adaptive Server is running. Therefore, using **xp_cmdshell** with the **xp_cmdshell context**

configuration parameter set to 0, any user can execute operating system commands using the permissions of the account running Adaptive Server. This account may have fewer restrictions than the user's own account.

For more information about t **xp_cmdshell context**, see the *System Administration Guide*

- Regardless of the value of **xp_cmdshell context**, if the user who is executing **xp_cmdshell** is not a System Administrator (does not have the **sa_role**), a System Administrator must have granted that user explicit permission to execute **xp_cmdshell**. For example, the following statement grants "joe" permission to execute **xp_cmdshell**:

  ```
  grant execute on xp_cmdshell to joe
  ```

Permissions

By default, only a System Administrator can execute **xp_cmdshell**. A System Administrator can grant execute permission to other users.

See Also

System procedures	**sp_configure**

xp_deletemail

(Windows NT only)

Function

Deletes a message from the Adaptive Server message inbox.

Syntax

```
xp_deletemail [msg_id]
```

Parameters

msg_id – is the message identifier of the mail message to be deleted.

Examples

1.
```
declare @cur_msg_id binary(255)
xp_deletemail @msg_id = @cur_msg_id
```

   Deletes from the Adaptive Server message inbox the message with the message identifier specified in the *cur_msg_id* variable.

2.
```
xp_deletemail
```

   Deletes the first message from the Adaptive Server message inbox.

Comments

- Obtain the *msg_id* using **xp_findnextmsg**.
- If the *msg_id* parameter is not used, the message to be deleted defaults to the first message in the message inbox.

Permissions

By default, only a System Administrator can execute **xp_deletemail**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_findnextmsg**, **xp_startmail**
System procedures	**sp_processmail**

xp_enumgroups

(Windows NT only)

Function

Displays groups for a specified Windows NT domain.

Syntax

xp_enumgroups [*domain_name*]

Parameters

domain_name – is the Windows NT domain for which you are listing user groups.

Examples

1. **xp_enumgroups**

   Lists all user groups on the Windows NT computer running XP Server.

2. **xp_enumgroups 'PCS'**

   Lists all user groups in the PCS domain.

Comments

- **xp_enumgroups** displays all local user groups if no parameter is passed.
- A **domain** is a named collection of computers that share a common user account database and security policy.
- A return status of 0 indicates success; 1 indicates failure.

Permissions

By default, only a System Administrator can execute **xp_enumgroups**. A System Administrator can grant this permission to other users.

xp_findnextmsg

(Windows NT only)

Function

Retrieves the next message identifier from the Adaptive Server message inbox.

Syntax

```
xp_findnextmsg @msg_id = @msg_id output [, type]
   [, unread_only = {true | false}]
```

Parameters

msg_id – on input, specifies the message identifier that immediately precedes the one you are trying to retrieve. Places the retrieved message identifier in the *msg_id* output parameter, which must be of type binary.

type – is the input message type based on the MAPI mail definition. The only supported message type is **CMC:IPM**. A NULL value or no value defaults to **CMC:IPM**.

unread_only – if this parameter is set to **true**, **xp_findnextmsg** considers only unread messages. If this parameter is set to **false**, **xp_findnextmsg** considers all messages, both read and unread, when retrieving the next message identifier. The default is **true**.

Examples

1. `xp_findnextmsg @msg_id = @out_msg_id output`

   Returns, in the *@out_msg_id* output variable, the message identifier of the next unread message after the message specified by the *@out_msg_id*.

2. `xp_findnextmsg @msg_id = @out_msg_id output, NULL, @unread_only = false`

   Returns, in the *@out_msg_id* output variable, the message identifier of the next message after the message specified by the *@out_msg_id*. The message may be read or unread.

Comments

- When **xp_findnextmsg** can find no more messages in the inbox, it returns a status of 1.

- **xp_deletemail** and **xp_readmail** use the message identifier returned by **xp_findnextmsg**.

Permissions

By default, only a System Administrator can execute **xp_findnextmsg**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_deletemail**, **xp_readmail**, **xp_startmail**
System procedures	**sp_processmail**

xp_logevent

(Windows NT only)

Function

Provides for logging a user-defined event in the Windows NT Event Log from within Adaptive Server.

Syntax

```
xp_logevent error_number, message [, type]
```

Parameters

error_number – is the user-assigned error number. It must be equal to or greater than 50000.

message – is the text of the message that is displayed in the description field of the event viewer. The maximum length of the message is 255 bytes. Enclose the message in quotes.

type – describes the urgency of the event. Values are **informational**, **warning**, and **error**. The default is **informational**. Enclose the value in quotes.

Examples

1. `xp_logevent 55555, 'Email message deleted.'`

   An informational event, number 55555, will be logged in the Windows NT Event Log. The text of the description in the event detail window is “Email message deleted”.

2. `xp_logevent 66666, 'DLL not found.', 'error'`

   An error event, number 66666, will be logged in the Windows NT Event Log. The text of the description in the event detail window is “DLL not found”.

Comments

- The following table describes the default event details for events generated with **xp_logevent**:

Detail	Value
User	N/A
Computer	Name of machine running XP Server
Event ID	12
Source	Name of Adaptive Server
Category	**User**

Permissions

Only a System Administrator can execute **xp_logevent**.

xp_readmail

(Windows NT only)

Function

Reads a message from the Adaptive Server message inbox.

Syntax

```
xp_readmail [msg_id]
   [, recipients output]
   [, sender output]
   [, date_received output]
   [, subject output]
   [, cc output]
   [, message output]
   [, attachments output]
   [, suppress_attach = {true | false}]
   [, peek = {true | false}]
   [, unread = {true | false}]
   [, msg_length output]
   [, bytes_to_skip [output]]
   [, type [output]]
```

Parameters

msg _id – specifies the message identifier of the message to be read by **xp_readmail**. If the *msg_id* parameter is not used, the message defaults to the first unread message in the message box, if **unread** is **true**, or to the first message in the message box, if **unread** is **false**.

recipients – is a semicolon-separated list of the recipients of the message.

sender – is the originator of the message.

date_received – is the date the message was received.

subject – is the subject header of the message.

cc – is a list of the message's copied (cc'd) recipients (separated by semicolons).

message – is the text of the message body. If the length of the message body, obtained from the *msg_length* output parameter, is greater than 255, use the *byte_to_skip* and *msg_length* parameters to read the message in 255-byte increments.

attachments – is a list of the temporary paths of the attachments (separated by semicolons). *attachments* is ignored if *suppress_attach* is **true**.

suppress_attach – if set to **true**, prevents the creation of temporary files for attachments. The default is **true**.

peek– if set to **false**, flags the message as unread after it has been read. If set to **true**, flags the message as an unread message, even after it has been read. The default is **false**.

unread_only – if set to **true**, **xp_readmail** considers only unread messages. If set to **false**, **xp_readmail** considers all messages, whether they are flagged as read or unread. The default is **true**.

msg_length – is the total length of the message, in bytes. Used with the *bytes_to_skip* parameter, allows **xp_readmail** to read messages in 255-byte increments.

bytes_to_skip – on input, if not 0, specifies the number of bytes to skip before reading the next 255 bytes of the message into the **message** output parameter. On output, contains the offset in the message (the previous value of *bytes_to_skip* plus the *msg_length* that is output with the call) from which to start reading the next 255-byte increment.

type – is the message type based on the MAPI mail definition. The only supported message type is **CMC:IPM**. A NULL value or no value defaults to **CMC:IPM**.

Examples

1.
```
declare @msgid binary(255)
declare @originator varchar(20)
declare @mess varchar(255)
exec xp_findnextmsg @msg_id = @msgid output
exec xp_readmail @msg_id = @msgid,
@sender = @originator output,
@message = @mess output
```

xp_readmail reads the first unread message in the message inbox. It gets the message identifier for this message from the *@msgid* variable, where it has been stored by the **xp_findnextmsg** ESP. **xp_readmail** stores the sender's name in the *@originator* variable and the message body in the *@mess* variable.

2.
```
declare @msgid binary(255)
declare @mess varchar(255)
declare @len int
declare @skip int = 0
exec xp_findnextmsg @msgid output
exec xp_readmail @msg_id = @msgid,
@message = @mess output
@msg_length = @len output,
@bytes_to_skip = @skip output
print @mess
if (@len > 255)
begin
    while (@skip < @len)
    begin
        xp_readmail @msg_id = @msgid,
        @message = @mess output,
        @bytes_to skip = @skip output
        print @mess
    end
end
```

   Reads the first 255 bytes of the message for which the message identifier is output by **xp_findnextmsg**. If the total length of the message exceeds 255 bytes, reads the next 255 bytes and continues until there are no more bytes to read.

Comments

- **xp_readmail** reads a message from the Adaptive Server message inbox.
- To get the message identifier of the next message in the message inbox, use **xp_findnextmsg**.

Permissions

By default, only a System Administrator can execute **xp_readmail**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_deletemail**, **xp_findnextmsg**, **xp_sendmail**, **xp_startmail**
System procedures	**sp_processmail**

xp_sendmail

(Windows NT only)

Function

Sends a message to the specified recipients. The message is either text or the results of a Transact-SQL query.

Syntax

```
xp_sendmail recipient [; recipient] . . .
   [, subject]
   [, cc_recipient] . . .
   [, bcc_recipient] . . .
   [, {query | message}]
   [, attachname]
   [, attach_result = {true | false}]
   [, echo_error = {true | false}]
   [, include_file [, include_file] . . .]
   [, no_column_header = {true | false}]
   [, width]
   [, separator]
   [, dbuser]
   [, dbname]
   [, type]
   [, include_query = {true | false}]
```

Parameters

recipient – is the email address of the user who will receive the message. At least one recipient is required. Separate multiple recipients with semicolons.

subject – is the optional message subject header. If not used, defaults to “Sybase SQL Server Message”.

cc_recipient – is a list of the message’s copied (cc’d) recipients (separated by semicolons).

bcc_recipient – is the list of the message’s blind- copied (bcc’d) recipients (separated by semicolons).

query – is one or more Transact-SQL statements. The results are sent to the recipients of the message. If *query* is used, *message* cannot be used.

message – is the text of the message being sent. If *message* is used, *query* cannot be used.

attachname – is the name of the file containing the results of a query, which is included as an attachment to the message, when the *query* parameter is used. If *attachname* is used, *attach_result* must be set to **true**. If *attach_result* is **true** and *attachname* is not specified, the prefix of the attached file's generated file name is "syb" followed by 5 random digits followed by the ".txt" extension; for example, *syb84840.txt*. This parameter is ignored if the *message* parameter is used.

attach_result – if set to **true**, sends the results of a query as an attachment to the message. If set to **false**, sends the results directly in the message body. The default is **false**. This parameter is ignored if the *message* parameter is used.

echo_error – if set to **true**, sends Adaptive Server messages, including the count of rows affected message, along with the query results. If set to **false**, does not send Adaptive Server messages. The default is **true**. This parameter is ignored if the *message* parameter is used.

include_file – is a list of files to be included as attachments to the message, separated by semicolons. The files can be specified as file names, path names, or relative path names and can be either text or binary files.

no_column_header – if set to **true**, column headers are sent with query results. If set to **false**, column headers are not sent. The default is **false**. This parameter is ignored if the *message* parameter is used.

no_output – if set to **true**, no output is sent to the session that sent the mail. If set to **false**, the session sending the mail receives output. The default is **false**. This parameter is ignored if the *message* parameter is used.

width – specifies, in characters, the width of the results sets when query results are sent in a message. *width* is the same as the **/w** option in **isql**. Result rows are broken by the newline character when the specified *width* is reached. The default is 80 characters. This parameter is ignored if the *message* parameter is used.

separator – specifies the character to be used as a column separator when query results are sent in a message. *separator* is the same as the **/s** option in **isql**. The default is the tab character. This parameter is ignored if the *message* parameter is used.

dbuser – specifies the database user name to be assumed for the user context for executing queries when the *query* parameter is used. The default is "guest." This parameter is ignored if the *message* parameter is used.

dname – specifies the database name to be assumed for the database context for executing queries when the *query* parameter is used. The default is "master." This parameter is ignored if the *message* parameter is used.

type – is the input message type based on the MAPI mail definition. The only supported message type is **CMC:IPM**. A NULL value or no value defaults to **CMC:IPM**.

include_query – if set to **true**, the query or queries used in the *query* parameter are appended to the results set. If set to **false**, the query is not appended. The default is **false**. *include_query* is ignored if the *message* parameter is used.

Examples

1. ```
   xp_sendmail @recipient = "sally";"ramon",
   @subject = "Adaptive Server Backup Status",
   @message = "Adaptive Server Backup for SERVER2 is
   complete.",
   @copy_recipient="admin"
   ```

   xp_sendmail sends a text message on the backup status of an Adaptive Server to "sally" and "ramon" with a copy to the "admin" group.

2. ```
   xp_sendmail "peter",
   @query = "select * from authors",
   @attachname = "au_list.res",
   @attach_result= true
   ```

   Sends "peter" the results of a query on the *authors* table. The results are in an attachment to the message, which consists of a file named *au_lis.res*, which is in the directory from which the server is being executed.

Comments

- The following parameters are related to the results of queries sent in a message when the *query* parameter is used. They are ignored if the *message* parameter is used instead: *attachname*, *attach_result*, *echo_error*, *no_column_header*, *no_output*, *width*, *separator*, *dbuser*, *dname*, *include_query*.

Permissions

By default, only a System Administrator can execute **xp_sendmail**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_deletemail**, **xp_findnextmsg**, **xp_readmail**, **xp_startmail**
System procedures	**sp_processmail**
Utility	**isql**

xp_startmail

(Windows NT only)

Function

Starts an Adaptive Server mail session.

Syntax

xp_startmail [*mail_user*] [, *mail_password*]

Parameters

mail_user – is a mail profile name used by Adaptive Server to log into the Windows NT mail system. If *mail_user* is not used, **xp_startmail** uses the mail user name that was used to set up Sybmail's Adaptive Server account.

mail_password – is the mail password used by Adaptive Server to log into the Windows NT mail system. If *mail_password* is not used, **xp_startmail** uses the mail password that was used to set up Sybmail's Adaptive Server account.

Examples

1. **xp_startmail**

   Starts an Adaptive Server mail session using the mail user name and password for Sybmail's user account.

2. **xp_startmail "mailuser", "tre55uu"**

   Starts an Adaptive Server mail session with "mailuser" as the profile name and the password associated with that profile name.

Comments

- **xp_startmail** will not start an Adaptive Server mail session if one is already running.
- An Adaptive Server mail session must be started, either by an explicit call to **xp_startmail** or by configuring Adaptive Server to start an Adaptive Server mail session automatically at start-up, before any Sybmail-related system ESPs or the **sp_processmail** stored procedure can be executed. For information about initiating an Adaptive Server mail session automatically at start-up, see **start mail session** in the *System Administration Guide.*

- When the Windows NT **automail** session is not on, you must use the *mail_user* and *mail_password* parameters with **xp_startmail**.
- To see the default *mail_user* value from the *fullname* field for the "sybmail" user account, use the **sp_displaylogin** system procedure as follows:

```
sp_displaylogin sybmail
```

Permissions

By default, only a System Administrator can execute **xp_startmail**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_stopmail**
System procedures	**sp_configure**

xp_stopmail

(Windows NT only)

Function

Stops an Adaptive Server mail session.

Syntax

```
xp_stopmail
```

Parameters

None.

Examples

1. `xp_stopmail`

Stops an Adaptive Server mail session.

Comments

- Sybmail-related system ESPs and the **sp_processmail** stored procedure cannot be executed after an Adaptive Server mail session has been terminated with **xp_stopmail**.

Permissions

By default, only a System Administrator can execute **xp_stopmail**. A System Administrator can grant this permission to other users.

See Also

System ESPs	**xp_startmail**
System procedures	**sp_processmail**

10 *dbcc* Stored Procedures

This chapter describes the **dbcc** stored procedures. These procedures access the tables only in the *dbccdb* database or in the alternate database, *dbccalt*. For details on setting up *dbccdb* or *dbccalt*, see the *System Administration Guide*. For information on the tables used in these databases, see Chapter 12, "dbccdb Tables," in the *Adaptive Server Reference Manual*.

Table 10-1 lists the **dbcc** stored procedures described in this chapter. For details on the **dbcc** system procedure **sp_plan_dbccdb**, see **sp_plan_dbccdb**. For more information on this system procedure and the **dbcc** stored procedures, see the *System Administration Guide*.

Table 10-1: dbcc stored procedures

Procedure Name	Description
sp_dbcc_alterws	Changes the size of the specified workspace to a specified value, and initializes the workspace.
sp_dbcc_configreport	Generates a report that describes the configuration information used by the **dbcc checkstorage** operation for the specified database.
sp_dbcc_createws	Creates a workspace of the specified type and size on the specified segment and database.
sp_dbcc_deletedb	Deletes from *dbccdb* all the information related to the specified target database.
sp_dbcc_deletehistory	Deletes the results of **dbcc checkstorage** operations performed on the target database before the specified date and time.
sp_dbcc_differentialreport	Generates a report that highlights the changes in I/O statistics and faults that took place between two **dbcc** operations
sp_dbcc_evaluatedb	Recomputes configuration information for the target database and compares it to the current configuration information.
sp_dbcc_faultreport	Generates a report covering fault statistics for the **dbcc checkstorage** operations performed for the specified object in the target database on the specified date.

Table 10-1: dbcc stored procedures (continued)

Procedure Name	Description
sp_dbcc_fullreport	Runs **sp_dbcc_summaryreport**, **sp_dbcc_configreport**, **sp_dbcc_statisticsreport**, and **sp_dbcc_faultreport short** for *database..object_name* on or before the specified *date*.
sp_dbcc_runcheck	Runs **dbcc checkstorage** on the specified database, then runs **sp_dbcc_summaryreport** or a report you specify
sp_dbcc_statisticsreport	Generates an allocation statistics report on the specified object in the target database.
sp_dbcc_summaryreport	Generates a summary report on the specified database.
sp_dbcc_updateconfig	Updates the *dbcc_config* table in *dbccdb* with the configuration information of the target database.

Specifying the Object Name and Date

Several **dbcc** stored procedures use parameters for the object name and date. This section provides important information on specifying the object name and date.

Specifying the Object Name

The object name specifies only the name of the table or index for which to generate a report. When you specify an object name, you must also specify a database name (*dbname*). You cannot specify an owner for the object. If the specified object name is not unique in the target database, the system procedure generates a report on all objects with the specified name.

Specifying the Date

Use the following syntax to specify the date and time (optional):

```
mm/dd/yy[:hh:mm:ss]
```

A 24-hour clock is assumed.

When you specify the date, the system procedures interpret it as follows:

- If both the date and the time are specified, the **dbcc** operation that completed at the specified date and time is selected for the report.
- If the specified date is the current date, and no time is specified, the time is automatically set to the current time. The **dbcc** operation that completed within the previous 24 hours with a finish time closest to the current time is selected for the report.
- If the specified date is not the current date, and no time is specified, the time is automatically set to "23:59:59". The **dbcc checkstorage** operation that completed with a finish date and time closest to the specified date and system-supplied time is selected for the report.

For example, suppose the most recent **dbcc checkstorage** operation completed on March 4, 1997 at 10:20:45.

If you specify the date as "03/04/97", the system procedure interprets the date as 03/04/97:23:59:59. This date and time are compared to the actual finish date and time of 03/04/97:10:20:45.

If you specify the date as "03/04/97:10:00:00", the operation that completes at 10:20:45 is not selected for the report because only the operations that complete on or before the specified time meet the criteria.

If you specify the date as "03/06/97", no report is generated because the most recent operation completed more than 24 hours earlier.

sp_dbcc_alterws

Function

Changes the size of the specified workspace to a specified value, and initializes the workspace.

Syntax

sp_dbcc_alterws *dbname*, *wsname*, "*wssize*[K|M]"

Parameters

dbname – is the name of the database in which the workspace resides. Specify either **dbccdb** and **dbccalt.**

wsname – specifies the name of the workspace to alter.

wssize – is the new size of the workspace, specified by **K** (kilobytes) or **M** (megabytes). If you do not specify **K** or **M**, *wssize* specifies the number of pages. Page size is platform-dependent. The minimum size for a workspace is 24 pages.

Example

1. sp_dbcc_alterws dbccdb, scan_ws_000001, "30M"

```
Workspace scan_ws_000001 has been altered
successfully to size 30MB
```

Changes the size of the *scan_ws_000001* workspace on *dbccdb* to 30MB.

Comments

- **sp_dbcc_alterws** changes the size of the specified workspace to the specified value and initializes the workspace.
- To achieve maximum performance, make sure you have configured a buffer pool of at least 16K before you alter a workspace.
- Use **sp_plan_dbccdb** to determine size estimates before altering the workspace.
- The workspace must exist before it can be altered. For information on creating workspaces, see **sp_dbcc_createws**.
- To delete a workspace, in *dbccdb* issue:

  drop table *workspace_name*

- For more information on the *scan* and *text* workspaces, and the *dbccalt* database, see the *System Administration Guide*.

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_alterws**.

Tables Used

master..sysdatabases, *syssegments*, *sysobjects*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_createws**, **sp_dbcc_evaluatedb**
System procedures	**sp_plan_dbccdb**, **sp_help**, **sp_helpdb**

sp_dbcc_configreport

Function

Generates a report that describes the configuration information used by the **dbcc checkstorage** operation for the specified database.

Syntax

sp_dbcc_configreport [*dbname*]

Parameters

dbname – specifies the name of the database. If *dbname* is not specified, the report contains information on all databases in *dbccdb..dbcc_operation_log*.

Examples

1. sp_dbcc_configreport

```
Reporting configuration information of database sybsystemprocs.

Parameter Name              Value                          Size

database name               sybsystemprocs                 51200K
dbcc named cache            default data cache             1024K
text workspace              textws_001 (id = 544004969)    128K
scan workspace              scanws_001 (id = 512004855)    1024K
max worker processes        1
operation sequence number    2
```

Generates a report on the configuration information related to **dbcc** for the *sybsystemprocs* database. The "Value" column lists the object name, where applicable, and the size.

Comments

- **sp_dbcc_configreport** generates a report that describes the configuration information used by **dbcc** operations for the specified database. This information is stored in the *dbcc_config* table.
- To evaluate the most current configuration parameters, run **sp_dbcc_updateconfig** before running **sp_dbcc_configreport**.
- To change the configuration values for a workspace, use **sp_dbcc_alterws**.

Permissions

Any user can run **sp_dbcc_configreport.**

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_config*

See Also

Commands	dbcc
dbcc stored procedures	**sp_dbcc_alterws**, **sp_dbcc_fullreport**, **sp_dbcc_statisticsreport**, **sp_dbcc_summaryreport**, **sp_dbcc_updateconfig**

sp_dbcc_createws

Function

Creates a workspace of the specified type and size on the specified segment and database.

Syntax

```
sp_dbcc_createws dbname, segname, [wsname], wstype,
   "wssize[K|M]"
```

Parameters

dbname – is the name of the database in which the workspace is to be created. Values are *dbccdb* and *dbccalt*.

segname – is the name of the segment for the workspace.

wsname – is the name of the workspace. If the value is null, **sp_dbcc_createws** generates the name *scan_ws_nnnnnn* for the *scan* workspace and *text_ws_nnnnnn* for the *text* workspace, where *nnnnnn* is a unique 6-digit number.

wstype – specifies the type of workspace to be create. Values are **scan** and **text**.

wssize – is the workspace size, specified with **K** (kilobytes) or **M** (megabytes). If you do not specify **K** or **M**, *wssize* specifies the number of pages. The minimum size for a workspace is 24 pages.

Example

1. ```
   sp_dbcc_createws dbccdb, scanseg, scan_ws_pubs2,
   scan, "10M"
   ```

   Creates a 10MB *scan* workspace named *scan_ws_pubs2* on the *scanseg* segment in *dbccdb*.

2. ```
   sp_dbcc_createws dbccdb, textseg, text, "14M"
   ```

   Creates a 14MB *scan* workspace named *text_ws_000001* on the *textseg* segment in *dbccdb*.

Comments

- **sp_dbcc_createws** creates a workspace with the specified name and size and initializes it.

- Before you create a workspace, create the segment with **sp_addsegment.**
- Before you create a workspace, make sure you have configured a buffer pool of at least 16K, to achieve maximum performance.
- Use **sp_plan_dbccdb** to determine size estimates.
- After creating a workspace, run **sp_dbcc_updateconfig** to record the new configuration information in *dbcc_config*.
- Each workspace must have a unique name.
- To delete a workspace, in *dbccdb* issue:

  ```
  drop table workspace_name
  ```

- For more information on the *scan* and *text* workspaces, see the *System Administration Guide.*
- For information on the *dbccalt* database, see the *System Administration Guide.*

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_createws.**

Tables Used

master..sysdatabases, *syssegments*, *sysobjects*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_alterws**, **sp_dbcc_evaluatedb**
System procedures	**sp_addsegment**, **sp_plan_dbccdb**, **sp_help**, **sp_helpsegment**

sp_dbcc_deletedb

Function

Deletes from *dbccdb* all the information related to the specified target database.

Syntax

sp_dbcc_deletedb [*dbname*]

Parameters

dbname – specifies the name of the target database for which you want the configuration information deleted. If you do not specify a value for *dbname*, Adaptive Server deletes data from all databases in *dbccdb..dbcc_config*. If the target database is *dbccdb*, and *dbccalt* exists, Adaptive Server deletes the data from *dbccalt*.

Example

1. sp_dbcc_deletedb "engdb"

```
All information for database engdb has been
deleted from dbccdb.
```

Deletes all information for the database named *engdb* from *dbccdb*.

Comments

- **sp_dbcc_deletedb** deletes from *dbccdb* all the information related to the specified target database, including configuration information and the results of previous **dbcc checkstorage** operations.
- If the deleted database is *dbccdb*, and the *dbccalt* database exists, **sp_dbcc_deletedb** deletes the configuration information and results of *dbccdb* from *dbccalt*.
- To remove the results of **dbcc checkstorage** operations created before a specific date, use **sp_dbcc_deletehistory**.
- For information about the *dbccalt* database, see the *System Administration Guide*.

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_deletedb**.

Tables Used

master..sysdatabases, *dbccdb..dbcc_config*, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_counters*, *dbccdb..dbcc_faults*, *dbccdb..dbcc_fault_params*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_deletehistory**, **sp_dbcc_evaluatedb**
System procedures	**sp_plan_dbccdb**

sp_dbcc_deletehistory

Function

Deletes the results of **dbcc checkstorage** operations performed on the target database before the specified date and time.

Syntax

```
sp_dbcc_deletehistory [cutoffdate [, dbname]]
```

Parameters

cutoffdate – deletes all entries made on or before this date. This parameter is of type *datetime*. If a date is not specified, only the results of the last operation are retained. For more information, see "Specifying the Date" on page 10-2.

dbname – specifies the name of the database for which the data must be deleted. If not specified, **sp_dbcc_deletehistory** deletes the history information for all databases in *dbccdb..dbcc_config*.

Examples

```
1. sp_dbcc_deletehistory "03/04/1997", "pubs2"
```

Deletes results of all operations performed on the database *pubs2* on or before March 4, 1997.

Comments

- **sp_dbcc_deletehistory** deletes the results of **dbcc checkstorage** operations performed on the target database before the specified date and time.
- If the target database is *dbccdb*, and the *dbccalt* database exists, **sp_dbcc_deletehistory** deletes historical data for *dbccdb* from *dbccalt*.
- The value specified for *cutoffdate* is compared to the finish time of each **dbcc** operation.
- To see the dates when **dbcc checkstorage** was run so that you can choose the value for *cutoffdate*, run **sp_dbcc_summaryreport**.
- For information on the *dbccalt* database, see the *System Administration Guide*.

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_deletehistory** on a specific database. Only a System Administrator can run **sp_dbcc_deletehistory** without specifying a database name.

Tables Used

master..sysdatabases, *master..sysdevices*, *master..sysusages*, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_counters*, *dbccdb..dbcc_faults*, *dbccdb..dbcc_fault_params*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_deletedb**, **sp_dbcc_evaluatedb**
System procedures	**sp_plan_dbccdb**

sp_dbcc_differentialreport

Function

Generates a report that highlights the changes in I/O statistics and faults that took place between two **dbcc** operations.

Syntax

```
sp_dbcc_differentialreport [dbname [, objectname]],
   [db_op] [, "date1" [, "date2"]]
```

Parameters

dbname – specifies the name of the database. If you do not specify a *dbname*, the report contains information on all databases in *dbccdb..dbcc_operation_log*.

objectname – specifies the name of the table or index for which you want the report generated. If *object_name* is not specified, statistics on all objects in the target database are reported.

db_op – specifies the source of the data to be used for the report. The only value is **checkstorage**. The report is generated on the data specified by *db_op* on *date1* and *date2* for the specified object in the target database. If dates are not specified, the last two operations of the type *db_op* are compared.

date1 – specifies the first date of a **dbcc checkstorage** operation to be compared.

date2 – specifies the last date of a **dbcc checkstorage** operation to be compared.

Examples

1.
```
sp_dbcc_differentialreport master, sysprocedures,
checkstorage, "05/01/97", "05/04/97"
```

   Generates a report that shows the changes in I/O statistics and faults that occurred in the *sysprocedures* table between May 1, 1997 and May 4, 1997

Comments

- **sp_dbcc_differentialreport** generates a report that highlights the changes in I/O statistics and faults that occurred between two **dbcc** operations. It compares counter values reported from two

instances of **dbcc checkstorage**. Only the values that have been changed are reported.

- If only one date is specified, the results of the **dbcc checkstorage** operation selected by the specified date are compared to the results of the **dbcc checkstorage** operation immediately preceding the selected operation.
- If no dates are specified, the results of last two **dbcc checkstorage** operations are compared.
- If **sp_dbcc_differentialreport** returns a number for *object_name*, it means the object was dropped after the **dbcc checkstorage** operation completed.
- If no changes occurred between the specified operations, **sp_dbcc_differentialreport** does not generate a report.

Permissions

Any user can run **sp_dbcc_differentialreport**.

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_counters*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_fullreport**, **sp_dbcc_statisticsreport**, **sp_dbcc_summaryreport**, **sp_dbcc_updateconfig**

sp_dbcc_evaluatedb

Function

Recomputes configuration information for the target database and compares it to the current configuration information.

Syntax

sp_dbcc_evaluatedb [*dbname*]

Parameters

dbname – specifies the name of the target database. If *dbname* is not specified, **sp_dbcc_evaluatedb** compares all databases listed in the *dbcc_config* table.

Example

1. sp_dbcc_evaluatedb

```
Recommended values for workspace size, cache size and worker
process count are:

Database name : sybsystemprocs
current scan workspace size : 400K      suggested scan workspace
size : 272K
current text workspace size : 208K      suggested text workspace
size : 208K
current cache size : 1024K      suggested cache size : 640K
current process count : 1        suggested process count : 1
```

Recomputes configuration information for the current database, *sybsystemprocs*, and suggests new values for some parameters.

Comments

- **sp_dbcc_evaluatedb** recomputes configuration information for the target database and compares the data to the current configuration information. It uses counter values recorded for the target database in the *dbcc_counters* table.
- The cache size is the size of the 16K buffer pool in the cache. For a 2K buffer pool, the minimum size of this cache must be the recommended value, plus 512.
- When the size and data distribution pattern of the target database changes, run **sp_dbcc_evaluatedb** to optimize the configuration information.

- To gather configuration information for the target database the first time, use **sp_plan_dbccdb**.
- To make sure you are evaluating the most current configuration parameters, run **sp_dbcc_updateconfig** before running **sp_dbcc_evaluatedb**.

Permissions

Only System Administrator or the Database Owner can run **sp_dbcc_evaluatedb**. Only a System Administrator can run **sp_dbcc_evaluatedb** without specifying a database name.

Tables Used

master..sysdatabases, *master..sysdevices*, *master..sysusages*, *dbccdb..dbcc_counters*, *dbccdb..dbcc_config*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_updateconfig**
System procedures	**sp_plan_dbccdb**

sp_dbcc_faultreport

Function

Generates a report covering fault statistics for the **dbcc checkstorage** operations performed for the specified object in the target database on the specified date.

Syntax

```
sp_dbcc_faultreport [report_type [, dbname
   [, objectname [, date ]]]]
```

Parameters

report_type – specifies the type of fault report. Valid values are **short** and **long**. The default is **short**.

dbname – specifies the name of the target database; for example, *master..sysdatabases*. If *dbname* is not specified, the report contains information on all databases in *dbccdb..dbcc_operation_log*.

object_name – specifies the name of the table or index for which you want the report generated. If *object_name* is not specified, statistics on all objects in the target database are reported.

date – specifies exact date and time that the **dbcc checkstorage** operation finished. You can find this value in *dbcc_operation_log.finish*. You can create the value by combining the date from *start time* and the hours and minutes from *end time* in the **sp_dbcc_summaryreport** output. If you do not specify *date*, Adaptive Server uses the date of the most recent operation.

Examples

1. sp_dbcc_faultreport "short"

```
Database Name : sybsystemprocs

 Table Name     Index  Type Code Description         Page Number
 -------------- ------ --------- ------------------- -----------
 sysprocedures        0    100031 page not allocated        5702
 sysprocedures        1    100031 page not allocated       14151
 syslogs              0    100022 chain start error        24315
 syslogs            0    100031 page not allocated     24315
```

Generates a short report of the faults found in tables in the *sybsystemprocs* database. The report includes the table name, the index number in which the fault occurred, the type code of the fault,

a brief description of the fault, and the page number on which the fault occurred.

2. sp_dbcc_faultreport "long"

```
Generating 'Fault Report' for object sysprocedures in database
sybsystemprocs.

Type Code: 100031; Soft fault, possibly spurious
Page reached by the chain is not allocated.
page id:  14151
page header:
0x0000374700003788000037460000005000648B803EF0001000103FE0080000F
Header for 14151, next 14216, previous 14150, id = 5:1
 time stamp = 0x0001000648B8, next row = 1007, level = 0
 free offset = 1022, minlen = 15, status = 128(0x0080)
.
.
.
```

Generates a long report of the faults found in tables in the *sybsystemprocs* database. This example shows the first part of the output of a long report. The complete report repeats the information for each object in the **target database** in which **dbcc checkstorage** found a fault. The data following the long string of numbers shown under the "page header" field (“Header for 14151, next 14216, previous 14150 ...”) describes the components of the "page header" string.

Comments

- **sp_dbcc_faultreport** generates a report that shows all faults for the specified object in the target database.
- If **sp_dbcc_faultreport** returns a number for *object_name*, it means the object was dropped after the **dbcc checkstorage** operation completed.
- For information on the fault ID, see the *type_code* column described in the *System Administration Guide.*
- For information on the fault status, see the *System Administration Guide.*

Permissions

Any user can run **sp_dbcc_faultreport.**

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_faults*, *dbccdb..dbcc_fault_params*

See Also

Commands	dbcc
dbcc stored procedures	sp_dbcc_fullreport, sp_dbcc_statisticsreport, sp_dbcc_summaryreport, sp_dbcc_updateconfig

sp_dbcc_fullreport

Function

Runs **sp_dbcc_summaryreport**, **sp_dbcc_configreport**, **sp_dbcc_statisticsreport**, and **sp_dbcc_faultreport short** for *database..object_name* on or before the specified *date*.

Syntax

```
sp_dbcc_fullreport [dbname [, objectname [, date]]]
```

Parameters

dbname – specifies the name of the database. If you do not specify *dbname*, the report contains information on all databases in *dbccdb..dbcc_operation_log*.

object_name – specifies the name of the table or index for which you want the report generated. If you do not specify *object_name*, statistics on all objects in the target database are reported.

date – specifies the date on which the **dbcc checkstorage** operation was performed. If you do not specify a *date*, the date of the last operation is used.

Examples

```
1. sp_dbcc_fullreport master, sysprocedures
```

Runs **sp_dbcc_summaryreport**, **sp_dbcc_configreport**, **sp_dbcc_statisticsreport**, and **sp_dbcc_faultreport short** for the most recent **dbcc checkstorage** operation run on the *sysprocedures* table in the *master* database.

Comments

- **sp_dbcc_fullreport** runs **sp_dbcc_summaryreport**, **sp_dbcc_configreport**, **sp_dbcc_statisticsreport**, and **sp_dbcc_faultreport short** for the specified database object on or before the specified date.

Permissions

Any user can run **sp_dbcc_fullreport**.

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_faults*, *dbccdb..dbcc_fault_params*, *dbccdb..dbcc_counters*

See Also

Commands	dbcc
dbcc stored procedures	**sp_dbcc_statisticsreport**, **sp_dbcc_summaryreport**, **sp_dbcc_updateconfig**

sp_dbcc_runcheck

Function

Runs **dbcc checkstorage** on the specified database, then runs **sp_dbcc_summaryreport** or a report you specify.

Syntax

```
sp_dbcc_runcheck dbname [, user_proc]
```

Parameters

dbname – specifies the name of the database on which the check is to be performed.

user_proc – specifies the name of the **dbcc** stored procedure or a user-created stored procedure that is to be run instead of **sp_dbcc_summaryreport**.

Example

1. ```
   sp_dbcc_runcheck "engdb"
   ```

   Checks the database *engdb* and generates a summary report on the information found.

2. ```
   sp_dbcc_runcheck "pubs2", sp_dbcc_fullreport
   ```

   Checks the database *pubs2* and generates a full report.

Comments

- **sp_dbcc_runcheck** runs **dbcc checkstorage** on the specified database.
- After the **dbcc checkstorage** operation is complete, **sp_dbcc_runcheck** runs **sp_dbcc_summaryreport** to generate a summary report. If you specify one of the other report-generating **dbcc** stored procedures for *dbcc_report*, **sp_dbcc_runcheck** runs that procedure instead of **sp_dbcc_summaryreport**. For a brief description and examples of all the report-generating stored procedures provided with *dbccdb*, see the *System Administration Guide*.
- You can write your own report-generating stored procedure and specify its name for *user_proc*. The stored procedure must be self-contained. **sp_dbcc_runcheck** cannot pass any parameters to Adaptive Server.

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_runcheck**.

Tables Used

master..sysdatabases, dbccdb..dbcc_config, dbccdb..dbcc_counters, dbccdb..dbcc_fault_params, dbccdb..dbcc_faults, dbccdb..dbcc_operation_log, dbccdb..dbcc_operation_results

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_summaryreport**

sp_dbcc_statisticsreport

Function

Generates an allocation statistics report on the specified object in the target database.

Syntax

sp_dbcc_statisticsreport [*dbname* [, *objectname* [, *date*]]]

Parameters

dbname – specifies the **target database**. If *dbname* is not specified, the report contains information on all databases in *dbccdb..dbcc_operation_log*.

objectname – specifies the name of the table or index for which you want the report generated. If you do not specify *objectname*, Adaptive Server reports statistics on all objects in the target database.

date – specifies the date on which the **dbcc checkstorage** operation was performed. If you do not specify *date*, Adaptive Server uses the date of the most recent operation.

Examples

1. sp_dbcc_statisticsreport 'sybsystemprocs', 'sysobjects'

```
Statistics Report on object sysobjects in database sybsystemprocs

 Parameter Name            Index Id Value
 ------------------------- -------- ------------
 count                     0        241.0
 max size                  0        99.0
 max count                 0        22.0
 bytes data                0        19180.0
 bytes used                0        22113.0
 count                     1        14.0
 max size                  1        9.0
 max level                 1        0.0
 max count                 1        14.0
 bytes data                1        56.0
 bytes used                1        158.0
 count                     2        245.0
```

```
max level                2        1.0
max size                 2        39.0
max count                2        71.0
bytes data               2        4377.0
bytes used               2        6995.0

 Parameter Name  Index Id  Partition  Value   Dev_name
---------------  --------  ---------  ------  -------------
page gaps               0          1    13.0  master
pages used              0          1    15.0  master
extents used            0          1     3.0  master
overflow pages          0          1     0.0  master
pages overhead          0          1     1.0  master
pages reserved          0          1     7.0  master
page extent gaps        0          1    11.0  master
ws buffer crosses       0          1     2.0  master
page extent crosses     0          1    11.0  master
pages used              1          1     2.0  master
extents used            1          1     1.0  master
overflow pages          1          1     0.0  master
pages overhead          1          1     1.0  master
pages reserved          1          1     6.0  master
page extent gaps        1          1     0.0  master
ws buffer crosses       1          1     0.0  master
page extent crosses     1          1     0.0  master
page gaps               2          1     4.0  master
pages used              2          1     6.0  master
extents used            2          1     1.0  master
overflow pages          2          1     0.0  master
pages overhead          2          1     1.0  master
pages reserved          2          1     2.0  master
page extent gaps        2          1     0.0  master
ws buffer crosses       2          1     0.0  master
page extent crosses      2         1     0.0  master
```

Generates a statistics report on the *sysobjects* table in the *master* database.

Comments

- **sp_dbcc_statisticsreport** generates an allocation statistics report on the specified object in the **target database**. It uses data from the *dbcc_counters* table, which stores information about page utilization and error statistics for every object in the target database.
- If **sp_dbcc_statisticsreport** returns a number for *object_name*, it means the object was dropped after the **dbcc checkstorage** operation completed.

- **sp_dbcc_statisticsreport** reports values recorded in the *dbcc_counters* table for the datatypes 5000–5019. See the *System Administration Guide*.

  For *bytes data*, *bytes used*, and *overflow pages*, **sp_dbcc_statisticsreport** reports the sum of the values reported for all partitions and devices.

  For *count*, *max count*, *max size* and *max level*, **sp_dbcc_statisticsreport** reports the largest of the values reported for all partitions and devices.

  sp_dbcc_statisticsreport reports information for each device and partition used by objects in the **target database** for the following rows:

  - *extents used*
  - *io errors*
  - *page gaps*
  - *page extent crosses*
  - *page extent gaps*
  - *page format errors*
  - *pages reserved*
  - *pages overhead*
  - *pages misallocated*
  - *pages not allocated*
  - *pages not referenced*
  - *pages used*

  The *page gaps*, *page extent crosses*, and *page extent gaps* indicate how the data pages for the objects are distributed on the database devices. Large values indicate less effectiveness in using larger buffer sizes and in data prefetch.

- If multiple **dbcc checkstorage** operations were run on a target database on the same day, **sp_dbcc_statisticsreport** generates a report based on the results of the last **dbcc checkstorage** operation that finished before the specified time.

Permissions

Any user can run **sp_dbcc_statisticsreport.**

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*, *dbccdb..dbcc_counters*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_fullreport**, **sp_dbcc_summaryreport**, **sp_dbcc_updateconfig**

sp_dbcc_summaryreport

Function

Generates a summary report on the specified database.

Syntax

sp_dbcc_summaryreport [*dbname* [, *op_name*]]

Parameters

dbname – specifies the name of the database for which you want the report generated. If you do not specify *dbname*, **sp_dbcc_summaryreport** generates reports on all databases in *dbccdb..dbcc_operation_log* for which the date is on or before the date and time specified by the *date* option.

op_name – specifies the operation. *optype* may be either **checkstorage**, which is the default, or **checkverify**, or both. If *op_name* is not specified, reports are generated for all operations.

Example

1. sp_dbcc_summaryreport

```
DBCC Operation : checkstorage

Database Name      Start time           End Time    Operation ID
    Hard Faults Soft Faults Text Columns Abort Count
       User Name
------------------ -------------------- ----------- ------------
    ---------- ---------- ------------ -----------
       -------------------------
sybsystemprocs     05/11/1999 14:53:11  14:53:32:163            1
             0           0            0           0
       sa
sybsystemprocs     05/11/1999 14:55:06  14:55:29:200            2
             0           0            0           0
       sa
sybsystemprocs    05/11/1999 14:56:10  14:56:27:750             3
             0           0            0           0
```

```
       sa

DBCC Operation : checkverify

Database Name      Start time           End Time    Operation ID
    Hard Faults    Soft Faults     User Name
------------------ -------------------- ----------- ------------
    -------------- -------------- --------------------
sybsystemprocs     05/11/1999 14:55:29  14:55:29:310            2
                 0              0 sa
```

Generates a summary report on the *master* database, providing information on all **dbcc checkstorage** and **dbcc checkverify** operations performed.

2. sp_dbcc_summaryreport "testdb"

```
DBCC Operation : checkstorage

Database Name      Start time           End Time    Operation ID
    Hard Faults Soft Faults Text Columns Abort Count
       User Name
------------------ -------------------- ------------ ------------
    ----------- ----------- ------------ -----------
       --------------------------
 testdb               05/11/1999 14:55:29  14:55:49:903            1
              0           0            0           0
       sa
 testdb                            05/11/1999 14:55:50
14:56:9:546             2
              0           0            0           0
       sa
 testdb                            05/11/1999 14:56:28
14:56:40:666            3
              0           0            0           0
       sa
```

Generates a summary report on the user database *testdb*, providing information on all **dbcc checkstorage** operations performed. **dbcc checkstorage** was the only operation run on this database, so no **dbcc checkverify** information appears on the report.

3. sp_dbcc_summaryreport null, "checkverify"

```
DBCC Operation : checkverify

Database Name      Start time           End Time    Operation ID
    Hard Faults    Soft Faults     User Name
------------------ -------------------- ----------- ------------
    -------------- -------------- --------------------
sybsystemprocs    05/11/1999 14:55:29   14:55:29:310          2
                 0              0 sa
```

Generates a summary report on the *master* database, providing information on all **dbcc checkverify** operations performed. Because **dbcc checkverify** was the specified operation, no **dbcc checkstorage** information appears on the report.

4. sp_dbcc_summaryreport sybsystemprocs, "checkstorage"

```
DBCC Operation : checkstorage

Database Name      Start time           End Time    Operation ID
    Hard Faults Soft Faults Text Columns Abort Count
       User Name
------------------ -------------------- ------------ ------------
    ----------- ----------- ------------ -----------
       -------------------------
sybsystemprocs     05/11/1999 14:53:11  14:53:32:163            1
              0           0            0           0
       sa
sybsystemprocs     05/11/1999 14:55:06  14:55:29:200            2
              0           0            0           0
       sa
sybsystemprocs     05/11/1999 14:56:10  14:56:27:750            3
              0           0            0           0
       sa
```

Generates a summary report on the *sybsystemprocs* database, providing information on all **dbcc checkstorage** operations performed. Because **dbcc checkstorage** was the specified operation, no **dbcc checkverify** information appears on the report.

Comments

- **sp_dbcc_summaryreport** generates a summary report of **checkstorage** or **checkverify** operations, or both, on the specified database.
- The report indicates the name of the database that was checked, the start and end time of the **dbcc checkstorage** run and the number of soft and hard faults found.
- The "Operation ID" column contains a number that identifies the results of each **dbcc checkstorage** operation on a given database at a specific time. The number provided in the report comes from the *opid* column of the *dbcc_operation_log* table. For more information, see the *System Administration Guide.*
- The "Text Columns" column shows the number of non-null text columns found by **dbcc checkstorage** during the run.
- The "Abort Count" column shows the number of tables that contained errors, which caused **dbcc checkstorage** to abort the check on the table. For details on the errors, run **sp_dbcc_faultreport.**

Permissions

Any user can run **sp_dbcc_summaryreport.**

Tables Used

master..sysdatabases, *dbccdb..dbcc_operation_log*, *dbccdb..dbcc_operation_results*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_fullreport**, **sp_dbcc_statisticsreport**, **sp_dbcc_updateconfig**

sp_dbcc_updateconfig

Function

Updates the *dbcc_config* table in *dbccdb* with the configuration information of the target database.

Syntax

```
sp_dbcc_updateconfig dbname, type, "str1" [, "str2"]
```

Parameters

dbname – is the name of the target database for which configuration information is being updated.

type – specifies the type name from the *dbcc_types* table. Table 10-2 on page 10-35 shows the valid values for *type*.

str1 – specifies the first configuration value for the specified *type* to be updated in the *dbcc_config* table. Table 10-2 on page 10-35 describes the expected value of *str1* for the specified *type*.

str2 – specifies the second configuration value for the specified *type* that you want to update in the *dbcc_config* table. Table 10-2 on page 10-35 describes the expected value of *str2* for the specified *type*.

Examples

1. **`sp_dbcc_updateconfig pubs2, "max worker processes", "4"`**

   Updates *dbcc_config* with the maximum number of worker processes for **dbcc checkstorage** to use when checking the *pubs2* database. The new maximum number of worker processes is 4.

2. **`sp_dbcc_updateconfig pubs2, "dbcc named cache", pubs2_cache, "10K"`**

   Updates *dbcc_config* with the size of the **dbcc** named cache "pubs2_cache". The new size is 10K.

3. **`sp_dbcc_updateconfig pubs2, "scan workspace", scan_pubs2`**

   Updates *dbcc_config* with the new name of the *scan* workspace for the *pubs2* database. The new name is *scan_pubs2*. This update is made after using **sp_dbcc_alterws** to change the name of the *scan* workspace.

4. **sp_dbcc_updateconfig pubs2, "text workspace", text_pubs2**

   Updates *dbcc_config* with the new name of the *text* workspace for the *pubs2* database. The new name is *text_pubs2*. This update is made after using **sp_dbcc_alterws** to change the name of the *text* workspace.

5. **sp_dbcc_updateconfig pubs2, "OAM count threshold", 5**

   Updates *dbcc_config* with the OAM count threshold value for the *pubs2* database. The new value is 5.

6. **sp_dbcc_updateconfig pubs2, "IO error abort", 3**

   Updates *dbcc_config* with the I/O error abort value for the *pubs2* database. The new value is 3.

7. **sp_dbcc_updateconfig pubs2, "linkage error abort", 8**

   Updates *dbcc_config* with the linkage error abort value for the *pubs2* database. The new value is 8.

Comments

- **sp_dbcc_updateconfig** updates the *dbcc_config* table for the **target database**.
- If the name of the target database is *dbccdb*, and the database *dbccalt* exists, **sp_dbcc_updateconfig** updates the *dbcc_config* table in *dbccalt*.
- If the target database name is not found in *dbcc_config*, **sp_dbcc_updateconfig** adds it and sets the operation sequence number to 0 before updating other configuration information.
- If the expected value for the specified *type* is a number, **sp_dbcc_updateconfig** converts the values you provide for *str1* and *str2* to numbers.

- Table 10-2 shows the valid type names to use for *type* and the expected value for *str1* or *str2*.

Table 10-2: Type names and expected values

type Name	Value expected for *str1* or *str2*
dbcc named cache	The name of the cache, specified by *str1*, and the new size (in kilobytes or megabytes) or the number of 2K pages, specified by *str2*.
IO error abort	The new error count, specified by *str1*. The value must be a number greater than 0. *str2* is not used with this type.
linkage error abort	The new linkage error count value specified in *str1*. The value must be a number greater than 0. *str2* is not used with this type.
max worker processes	The new number of worker processes, specified by *str1*. The value must be a number greater than 0. *str2* is not used with this type.
OAM count threshold	The new threshold count, specified by *str1*. The value must be a number greater than 0. *str2* is not used with this type.
scan workspace	The new name for the *scan* workspace, specified by *str1*. *str2* is not used with this type.
text workspace	The new name of the *text* workspace, specified by *str1*. *str2* is not used with this type.

- For more information on the *type* names and values, see the *System Administration Guide*.

Permissions

Only a System Administrator or the Database Owner can run **sp_dbcc_updateconfig**.

Tables Used

master..sysdatabases, *dbccdb..dbcc_types*, *dbccdb..dbcc_config*

See Also

Commands	**dbcc**
dbcc stored procedures	**sp_dbcc_evaluatedb**
System procedures	**sp_plan_dbccdb**, **sp_help**

For the Index, see volume 4, “Tables and Reference Manual Index.”

Volume 4, “Tables and Reference Manual Index,” contains the index entries for all volumes of the *Adaptive Server Reference Manual*.

For the Index, see volume 4, “Tables and Reference Manual Index.”

Sybase® Adaptive Server™ Enterprise Reference Manual

Volume 4: Tables and Reference Manual Index

Adaptive Server Enterprise Version 12
Document ID: 36274-01-1200-02
Last Revised: October 1999

Principal author: Enterprise Data Studios Publications

Contributing authors: Anneli Meyer, Evelyn Wheeler

Document ID: 36274-01-1200

This publication pertains to Adaptive Server Enterprise Version 12 of the Sybase database management software and to any subsequent version until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Sybase, the SYBASE logo, Adaptive Server, APT-FORMS, Certified SYBASE Professional, the Certified SYBASE Professional logo, Column Design, ComponentPack, Data Workbench, First Impression, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, Transact-SQL, Visual Components, VisualWriter, and VQL are registered trademarks of Sybase, Inc.

Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server Enterprise Monitor, Adaptive Warehouse, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, EnterpriseConnect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript,

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

12. *dbccdb* Tables

Index

About This Book

The *Adaptive Server Reference Manual* is a four-volume guide to Sybase® Adaptive Server™ Enterprise and the Transact-SQL® language.

Volume 1, "*Building Blocks*," describes the "parts" of Transact-SQL: datatypes, built-in functions, expressions and identifiers, SQLSTATE errors, and reserved words. Before you can use Transact-SQL sucessfully, you need to understand the purpose of each of these building blocks and how its use affects the results of Transact-SQL statements.

Volume 2, "*Commands*," provides reference information about the Transact-SQL commands, which you use to create statements.

Volume 3, "*Procedures*" provides reference information about system procedures, catalog stored procedures, extended stored procedures, and **dbcc** stored procedures. All procedures are created using Transact-SQL statements.

Volume 4, "*Tables and Reference Manual Index*," provides reference information about the system tables, which store information about your server, databases, users, and other information. It also provides information about the tables in the *dbccdb* and *dbccalt* databases. It also contains an index that covers the topics of all four volumes.

For information about the intended audience of this book, related documents, other sources of information, conventions used in this manual, and help, see "About This Book" in Volume 1.

How to Use This Book

This manual contains:

- Chapter 11, "System Tables," which provides information about all system tables in the *master* database, the auditing database, and any user databases (such as *pubs2*).
- Chapter 12, "dbccdb Tables," which provides information about the tables in the *dbccdb* and *dbccalt* databases.
- The Index, which provides entries for all four volumes of the *Adaptive Server Reference Manual*.

11 System Tables

Locations of System Tables

System tables may be located in:

- The *master* database,
- The *sybsecurity* database,
- The *sybsystemdb* database, or
- All databases.

Most tables in the *master* database are system tables. Some of these tables also occur in user databases—they are automatically created when the **create database** command is issued.

System Tables in *master*

The following system tables occur in the *master* database **only**:

System Table	Contents
syscharsets	One row for each character set or sort order
sysconfigures	One row for each configuration parameter that can be set by users
syscurconfigs	Information about configuration parameters currently being used by Adaptive Server
sysdatabases	One row for each database on Adaptive Server
sysdevices	One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases
sysengines	One row for each Adaptive Server engine currently online
syslanguages	One row for each language (except U.S. English) known to the server
syslisteners	One row for each type of network connection used by current Adaptive Server
syslocks	Information about active locks
sysloginroles	One row for each server login that possesses a system role
syslogins	One row for each valid Adaptive Server user account

System Table	Contents
syslogshold	Information about the oldest active transaction and the Replication Server® truncation point for each database
sysmessages	One row for each system error or warning
sysmonitors	One row for each monitor counter
sysprocesses	Information about server processes
sysremotelogins	One row for each remote user
sysresourcelimits	One row for each resource limit
syssecmechs	Information about the security services available for each security mechanism that is available to Adaptive Server
sysservers	One row for each remote Adaptive Server
syssessions	Only used when Adaptive Server is configured for Sybase's Failover in a high availability system. *syssessions* contains one row for each client that connects to Adaptive Server with the failover property (for example, **isql -Q**).
syssrvroles	One row for each server-wide role
systimeranges	One row for each named time range
systransactions	One row for each transaction
sysusages	One row for each disk piece allocated to a database

System Tables in *sybsecurity*

The following system tables occur in the *sybsecurity* database **only**:

System Table	Contents
sysauditoptions	One row for each global audit option
sysaudits_01 – sysaudits_08	The audit trail. Each audit table contains one row for each audit record.

System Tables in *sybsystemdb*

The following system tables occur in the *sybsystemdb* database **only**:

System Table	Contents
syscoordinations	One row for each remote participant of a distributed transaction

System Tables in All Databases

The following system tables occur in all databases:

System Table	Contents
sysalternates	One row for each Adaptive Server user mapped to a database user
sysattributes	One row for each object attribute definition.
syscolumns	One row for each column in a table or view, and for each parameter in a procedure
syscomments	One or more rows for each view, rule, default, trigger, and procedure, giving SQL definition statement
sysconstraints	One row for each referential and check constraint associated with a table or column
sysdepends	One row for each procedure, view, or table that is referenced by a procedure, view, or trigger
sysgams	Allocation bitmaps for an entire database
sysindexes	One row for each clustered or nonclustered index, one row for each table with no indexes, and an additional row for each table containing text or image data
sysjars	One row for each Java archive (JAR) file that is retained in the database. Uses row-level locking.
syskeys	One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server)
syslogs	Transaction log
sysobjects	One row for each table, view, procedure, rule, trigger default, log, and (in *tempdb* only) temporary object
syspartitions	One row for each partition (page chain) of a partitioned table

System Table	Contents
sysprocedures	One row for each view, rule, default, trigger, and procedure, giving internal definition
sysprotects	User permissions information
sysqueryplans	Abstract query plans and SQL text.
sysreferences	One row for each referential integrity constraint declared on a table or column
sysroles	Maps server-wide roles to local database groups
syssegments	One row for each segment (named collection of disk pieces)
sysstatistics	One or more rows for each indexed column on a user table. May also contain rows for unindexed column
systabstats	One row for each table, plus one row for each nonclustered index
systhresholds	One row for each threshold defined for the database
systypes	One row for each system-supplied and user-defined datatype
sysusermessages	One row for each user-defined message
sysusers	One row for each user allowed in the database
sysxtypes	One row for each extended, Java-SQL datatype. Uses row-level locking.

About the *sybdiagdb* Database

Sybase Technical Support may create the *sybdiagdb* database on your system for debugging purposes. This database holds diagnostic configuration data for use by Technical Support representatives. It should not be used by customers.

Rules for Using System Tables

This section describes rules, restrictions and usage information for system tables.

Permissions on System Tables

Permissions for use of the system tables can be controlled by the database owner, just like permissions on any other tables. By default,

when Adaptive Server is installed, the *installmodel* script grants **select** access to "public" (all users) for most system tables and for most fields in the tables. However, no access is given for some system tables, such as *systhresholds*, and no access is given for certain fields in other system tables. For example, all users, by default, can select all columns of *sysobjects* except *audflags*. To determine the current permissions for a particular system table, execute:

```
sp_helprotect system_table_name
```

For example, to check the permissions of *systhresholds* in *your_database*, execute:

```
use your_database
go
sp_helprotect systhresholds
go
```

Locking Schemes Used for System Tables

Unless noted otherwise, system tables use allpages locking.

Reserved Columns

The word "reserved" in the column description means that the column is not currently used by Adaptive Server.

Updating System Tables

All direct updates on system tables are by default not allowed — even for the database owner. Instead, Adaptive Server supplies system procedures to make any normally needed updates and additions to system tables.

You can allow direct updates to the system tables if it becomes necessary to modify them in a way that cannot be accomplished with a system procedure. To accomplish this, a System Security Officer must reset the configuration parameter called **allow updates to system tables** with the system procedure **sp_configure**. For more information, see the *System Administration Guide*.

There are entries in some of the *master* database tables that should not be altered by any user under any circumstances. For example, do not attempt to modify *syslogs* with a **delete**, **update**, or **insert** command. In addition, an attempt to delete all rows from *syslogs* will put

Adaptive Server into an infinite loop that eventually fills up the entire database.

Triggers on System Tables

You cannot create triggers on system tables. If you try to create a trigger on a system table, Adaptive Server returns an error message and cancels the trigger.

Aggregate Functions and Virtual Tables

Aggregate functions cannot be used on virtual tables such as *syslocks* and *sysprocesses*.

sysalternates

(all databases)

Description

sysalternates contains one row for each Adaptive Server user mapped (or aliased) to a user of the current database. When a user tries to access a database, Adaptive Server looks for a valid *uid* entry in *sysusers*. If none is found, it looks in *sysalternates.suid*. If the user's *suid* is found there, he or she is treated as the database user whose *suid* is listed in *sysalternates.altsuid*.

On the Adaptive Server distribution media, there are no entries in *sysalternates*.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID of user being mapped
altsuid	*smallint*	Server user ID of user to whom another user is mapped

Indexes

Unique clustered index on *suid*

sysattributes

(all databases)

Description

System attributes define properties of objects such as databases, tables, indexes, users, logins, and procedures. *sysattributes* contains one row for each of an object's attribute definitions (configured by various system procedures). *master..sysattributes* defines the complete set of valid attribute values and classes for Adaptive Server as a whole. It also stores attribute definitions for server-wide objects, such as databases and logins.

sysattributes should only be accessed indirectly using system procedures. The permissions required for modifying *sysattributes* depend on the system procedure you use.

Columns

Name	Datatype	Description
class	*smallint*	The attribute class ID. This describes the category of the attribute. In *master..sysattributes*, the special class 1 identifies all valid attributes for Adaptive Server. Class 0 identifies valid **classes** of attributes.
attribute	*smallint*	The attribute ID.
object_type	*char(2)*	The one- or two-letter character ID that defines the type of object to associate with the attribute.
object_cinfo	*varchar(30)*	A string identifier for the object (for example, the name of an application). This field is not used by all attributes.
object	*int null*	The object identifier. This may be an object ID, user ID, or database ID, depending on the type of object. If the object is a part of a table (for example, an index), then this column contains the object ID of the associated table.
object_info1	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.

Name	Datatype	Description
object_info2	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.
object_info3	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.
int_value	*int null*	An integer value for the attribute (for example, the display level of a user).
char_value	*varchar(255)*	A character value for the attribute (for example, a cache name).
text_value	*text null*	A text value for the attribute.
image_value	*image null*	An image value for the attribute.
comments	*varchar(255)*	Comments or additional information about the attribute definition.

Table 11-1 describes the *object_type* values and their meanings:

Table 11-1: Object types for attributes

ID	Object Type
D	Database
EL	External Login (for Component Integration Services)
I	Index
L	Login name
OD	Object Definition (for Component Integration Services)
P	Procedure
T	Table
TP	Text Page (for Component Integration Services)
U	Username
UI	Upgrade Item (used internally during user database upgrades)

Indexes

Unique clustered index on *class*, *attribute*, *object_type*, *object*, *object_info1*, *object_info2*, *object_info3*, *object_cinfo*

Nonclustered index on *object_type*, *object*, *object_info1*, *object_info2*, *object_info3*, *object_cinfo*

sysauditoptions

(*sybsecurity* database)

Description

sysauditoptions contains one row for each server-wide audit option and indicates the current setting for that option. Other types of auditing option settings are stored in other tables. For example, database-specific option settings are stored in *sysdatabases*, and object-specific option settings are stored in *sysobjects*. The default value for each option is 0, or "off." *sysauditoptions* can be accessed only by System Security Officers.

Columns

Name	Datatype	Description
num	*smallint*	Number of the server-wide option.
val	*smallint*	Current value; one of the following: 0 = **off** 1 = **pass** 2 = **fail** 3 = **on**
minval	*smallint*	Minimum valid value for this option.
maxval	*smallint*	Maximum valid value for this option.
name	*varchar(30)*	Name of option.
sval	*varchar(30)*	String equivalent of the current value: for example, "on", "off", "nonfatal".
comment	*varchar(255)*	Description of option.

sysaudits_01 – sysaudits_08

(*sybsecurity* database)

Description

These system tables contain the audit trail. Only one table at a time is active. The active table is determined by the value of the **current audit table** configuration parameter. An installation can have up to eight audit tables. For example, if your installation has three audit tables, the tables are named *sysaudits_01*, *sysaudits_02*, and *sysaudits_03*. An audit table contains one row for each audit record.

Columns

Name	Datatype	Description
event	*smallint*	Type of event being audited. See Table 11-3 on page -13.
eventmod	*smallint*	Further information about the event. Possible values are: 0 = no modifier for this event 1 = the event passed permission checking 2 = the event failed permission checking
spid	*smallint*	Server process ID of the process that caused the audit record to be written.
eventtime	*datetime*	Date and time of the audited event.
sequence	*smallint*	Sequence number of the record within a single event; some events require more than one audit record.
suid	*smallint*	Server login ID of the user who performed the audited event.
dbid	*int null*	Database ID in which the audited event occurred or the object / stored procedure / trigger resides, depending on the type of event.
objid	*int null*	ID of the accessed object or stored procedure / trigger.
xactid	*binary(6) null*	ID of the transaction containing the audited event. For a multi-database transaction, this is the transaction ID from the database where the transaction originated.
loginname	*varchar(30) null*	Login name corresponding to the *suid.*
dbname	*varchar(30) null*	Database name corresponding to the *dbid.*

Name	Datatype	Description
objname	*varchar(30) null*	Object name corresponding to the *objid*.
objowner	*varchar(30) null*	Name of the owner of *objid*.
extrainfo	*varchar(255) null*	Additional information about the audited event. This field contains a sequence of items separated by semicolons. See Table 11-2.

The *extrainfo* column contains a sequence of items separated by semicolons. Table 11-2 lists the items in the *extrainfo* column:

Table 11-2: Items in the extrainfo field

Item	Contents
Roles	Lists the roles that are active. The roles are separated by blanks.
Subcommand	The name of the subcommand or command option that was used for the event. For example, for the **alter table** command, the options "add column" or "drop constraint" might be used. Multiple subcommands or options are separated by commas.
Previous value	The value prior to the update if the event resulted in the update of a value.
Current value	The new value if the event resulted in the update of a value.
Other information	Additional security-relevant information that is recorded for the event.
Proxy information	The original login name, if the event occurred while a **set proxy** was in effect.
Principal information	The principal name from the underlying security mechanism, if the user's login is the secure default login, and the user logged into Adaptive Server via unified login. The value of this field is NULL, if the secure default login is not being used.

An example of an *extrainfo* column for the security-relevant event of changing an auditing configuration parameter might be:

```
sso_role;suspend auditing when full;1;0;;;;
```

This *extrainfo* column indicates that a System Security Officer changed the configuration parameter **suspend auditing when full** from 1 (suspend all processes that involve an auditing event) to 0 (truncate

the next audit table and make it the current audit table). The other columns in the audit record give other pertinent information. For example, the record contains the server user id (*suid*) and the login name (*loginname*).

The *event* column values that pertain to each audit event are listed in Table 11-3.

Table 11-3: Values in event and extrainfo column

Audit Option	*event*	Command or Access Audited	*extrainfo*
adhoc	1	User-defined audit record	*extrainfo* is filled by the *text* parameter of **sp_addauditrecord**
alter	2	**alter database**	**Roles:** Current active roles **Subcommand:** "ALTER SIZE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	3	**alter table**	**Roles:** Current active roles **Subcommand:** "ADD COLUMN", "REPLACE COLUMN", "ADD CONSTRAINT", or "DROP CONSTRAINT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
bcp	4	**bcp in**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
bind	6	**sp_bindefault**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Name of default **Proxy information:** Original login name, if a **set proxy** is in effect
	7	**sp_bindmsg**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Message ID **Proxy information:** Original login name, if a **set proxy** is in effect
	8	**sp_bindrule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Name of the rule **Proxy information:** Original login name, if a **set proxy** is in effect
create	9	**create database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	10	**create table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	11	**create procedure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
create (continued)	12	**create trigger**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	13	**create rule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	14	**create default**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	15	**sp_addmessage**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Message Number **Proxy information:** Original login name, if a **set proxy** is in effect
	16	**create view**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
dbaccess	17	Any access to the database by any user	**Roles:** Current active roles **Subcommand:** "USE CMD" or "OUTSIDE REFERENCE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
dbcc	81	**dbcc**	**Roles:** Current active roles **Subcommand:** The **dbcc** subcommand name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
delete	18	**delete** from a table	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	19	**delete** from a view	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
disk	20	**disk init**	**Roles:** Current active roles **Subcommand:** "disk init" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	21	**disk refit**	**Roles:** Current active roles **Subcommand:** "disk refit" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
disk (continued)	22	**disk reinit**	**Roles:** Current active roles **Subcommand:** "disk reinit" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	23	**disk mirror**	**Roles:** Current active roles **Subcommand:** "disk mirror" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	24	**disk unmirror**	**Roles:** Current active roles **Subcommand:** "disk unmirror" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	25	**disk remirror**	**Roles:** Current active roles **Subcommand:** "disk remirror" **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
drop	26	**drop database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	27	**drop table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
drop (continued)	28	**drop procedure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	29	**drop trigger**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	30	**drop rule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	31	**drop default**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	32	**sp_dropmessage**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Message number **Proxy information:** Original login name, if a **set proxy** is in effect
	33	**drop view**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
dump	34	**dump database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	35	**dump transaction**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
errors	36	Fatal error	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Error number.Severity.State **Proxy information:** Original login name, if a **set proxy** is in effect
	37	Non-fatal error	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Error number.Severity.State **Proxy information:** Original login name, if a **set proxy** is in effect
exec_procedure	38	Execution of a procedure	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** All input parameters **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
exec_trigger	39	Execution of a trigger	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
func_obj_access, **func_dbaccess**	85	Accesses to objects and databases via Transact-SQL functions	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
grant	40	**grant**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
insert	41	**insert** into a table	**Roles:** Current active roles **Subcommand:** If **insert**: "INSERT" If **select into**: "INSERT INTO" followed by the fully qualified object name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	42	**insert** into a view	**Roles:** Current active roles **Subcommand:** "INSERT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
load	43	**load database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	44	**load transaction**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
login	45	Any login to Adaptive Server	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Host name of the machine from which login was done **Proxy information:** Original login name, if a **set proxy** is in effect
logout	46	Any logouts from Adaptive Server	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Host name of the machine from which login was done **Proxy information:** Original login name, if a **set proxy** is in effect
revoke	47	**revoke**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
rpc	48	Remote procedure call from another server	**Roles:** Current active roles **Subcommand:** Name of client program **Previous value:** NULL **Current value:** NULL **Other information:** Server name, host name of the machine from which the RPC was done. **Proxy information**: Original login name, if a **set proxy** is in effect
	49	Remote procedure call to another server	**Roles:** Current active roles **Subcommand:** Procedure name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information**: Original login name, if a **set proxy** is in effect
security	50	Server start	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** **-d***masterdevicename* **-i***interfaces file path* **-S***servername* **-e***errorfilename* **Proxy information**: Original login name, if a **set proxy** is in effect
	51	Server shutdown	**Roles:** Current active roles **Subcommand:** "shutdown" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information**: Original login name, if a **set proxy** is in effect
	55	Role toggling	**Roles:** Current active roles **Subcommand:** NULL **Previous Value:** "on" or "off" **Current Value:** "on" or "off" **Other Information:** Name of the role being set **Proxy information**: Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
security (continued)	82	**sp_configure**	**Roles:** Current active roles **Subcommand:** Name of the configuration parameter **Previous Value:** The old parameter value if the command is setting a new value **Current Value:** The new parameter value if the command is setting a new value **Other Information:** Number of configuration parameter, if a parameter is being set; Name of the configuration file, if a configuration file is being used to set parameters **Proxy information:** Original login name, if a **set proxy** is in effect
	83	**online database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	76	Regeneration of a password by a System Security Officer (SSO)	**Roles:** Current active roles **Subcommand:** Setting SSO password **Previous value:** NULL **Current value:** NULL **Other information:** Login name **Proxy information:** Original login name, if a **set proxy** is in effect
	80	**proc_role** within a system procedure	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Required roles **Proxy information:** Original login name, if a **set proxy** is in effect
	85	**valid_user**	**Roles:** Current active roles **Subcommand:** "valid_user" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
security (continued)	88	**set proxy** or **set session authorization**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** Previous *suid* **Current value:** New *suid* **Other information:** NULL **Proxy information**: Original login name, if **set proxy** or **set session authorization** had no parameters; otherwise, NULL.
select	62	**select** from a table	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	63	**select** from a view	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information**: Original login name, if a **set proxy** is in effect
setuser	84	**setuser**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the user being set **Proxy information**: Original login name, if a **set proxy** is in effect
table_access	62	**select**	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information**: Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
table_access (continued)	18	**delete**	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	70	**update**	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	41	**insert**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
truncate	64	**truncate table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
unbind	67	**sp_unbindefault**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	68	**sp_unbindrule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
unbind (continued)	69	**sp_unbindmsg**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
update	70	**update** to a table	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	71	**update** to a view	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
view_access	63	**select**	**Roles:** Current active roles **Subcommand:** "SELECT INTO" "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	19	**delete**	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 11-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
view_access (continued)	42	**insert**	**Roles:** Current active roles **Subcommand:** "INSERT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	71	**update**	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
Note: This event is audited automatically. It is not controlled by an audit option.	73	Turning the **auditing** parameter on with **sp_configure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
Note: This event is audited automatically. It is not controlled by an audit option.	74	Turning the **auditing** parameter off with **sp_configure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

syscharsets

(*master* database only)

Description

syscharsets contains one row for each character set and sort order defined for use by Adaptive Server. One of the sort orders is marked in *master..sysconfigures* as the default sort order, which is the only one actually in use.

Columns

Name	Datatype	Description
type	*smallint*	The type of entity this row represents. Numbers from 1001 to 1999 represent character sets. Numbers from 2000 to 2999 represent sort orders.
id	*tinyint*	The ID for a character set or sort order. A sort order is defined by the combination of the sort order ID and the character set ID (*csid*). The character set is defined by *id*, which must be unique. Sybase reserves ID numbers 0–200.
csid	*tinyint*	If the row represents a character set, this field is unused. If the row represents a sort order, this is the ID of the character set that sort order is built on. A character set row with this ID must exist in this table.
status	*smallint*	Internal system status information bits.
name	*varchar(30)*	A unique name for the character set or sort order. Must contain only the 7-bit ASCII letters A-Z or a-z, digits 0-9, and underscores (_), and begin with a letter.
description	*varchar(255)*	An optional description of the features of the character set or sort order.
definition	*image*	The internal definition of the character set or sort order. The structure of the data in this field depends on the *type*.
sortfile	*varchar(30)*	The name of the sort order file.

Indexes

Unique clustered index on *id, csid, type*
Unique nonclustered index on *name*

syscolumns

(all databases)

Description

syscolumns contains one row for every column in every table and view, and a row for each parameter in a procedure.

Columns

Name	Datatype	Description
id	*int*	ID of table to which this column belongs or of procedure with which this parameter is associated
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
colid	*tinyint*	Column ID
status	*tinyint*	Bits 0–2 (values 1, 2, and 4) indicate bit positioning if the column uses the *bit* datatype. If the column uses the *text/image* datatype, bits 0 and 1 indicate replication status as follows: 01 = always replicate 10 = replicate only if changed 00 = never replicate Bit 3 (value 8) indicates whether NULL values are legal in this column. Bit 4 (value 16) indicates whether more than one check constraint exists for the column. Bits 5 and 6 are used internally. Bit 7 (value 128) indicates an identity column. Bit 8 is unused.
type	*tinyint*	Physical storage type; copied from *systypes*
length	*tinyint*	Physical length of data; copied from *systypes* or supplied by user
offset	*smallint*	Offset into the row where this column appears; if negative, this is a variable-length column
usertype	*smallint*	User type ID; copied from *systypes*

Name	Datatype	Description
cdefault	*int*	ID of the procedure that generates default value for this column
domain	*int*	Constraint ID of the first rule or check constraint for this column
name	*sysname*	Column name
printfmt	*varchar(255)*	Reserved
prec	*tinyint*	Number of significant digits
scale	*tinyint*	Number of digits to the right of the decimal point
remote_type	*int*	Maps local names to remote names. Required by the access methods of Component Integration Services to allow the software to pass native column datatype information in parameters to servers of class *access_server*.
remote_name	*varchar(30)*	Maps local names to remote names. Required by the access methods of Component Integration Services to construct a query using the proper column names for a remote table.
xtype	*int*	ID of the class. Used if a column in a table or a parameter in a procedure has a Java class as its datatype. When used, fields are not NULL, and the value of *type* is 0x39. Refer to *Java in Adaptive Server Enterprise* for more information.
xdbid	*int*	The database ID of the class. For system classes, the value is -1. Otherwise, the value is the current database ID. Used if a column in a table or a parameter in a procedure has a Java class as its datatype. Fields are not NULL, and the value of *type* is 0x39. Refer to *Java in Adaptive Server Enterprise* for more information.

Indexes

Unique clustered index on *id, number, colid*

syscomments

(all databases)

Description

syscomments contains entries for each view, rule, default, trigger, table constraint, and procedure. The *text* column contains the original definition statements. If the *text* column is longer than 255 bytes, the entries will span rows. Each object can occupy up to 65,025 rows.

Columns

Name	Datatype	Description
id	*int*	Object ID to which this text applies
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
colid	*tinyint*	Sequence of 255 rows for the object
texttype	*smallint*	0 for system-supplied comment (for views, rules, defaults, triggers, and procedures); 1 for user-supplied comment (users can add entries that describe an object or column)
language	*smallint*	Reserved
text	*varchar(255)*	Actual text of SQL definition statement
colid2	*tinyint*	Indicates next sequence of rows for the object (see *colid* above); object can have up to 255 sequences of 255 rows each
status	*smallint*	

➤ ***Note***

Do not delete the definition statements from the text column of *syscomments*. These statements are required for the Adaptive Server upgrade process. To encrypt a definition statement, run the system procedure **sp_hidetext**. To see if a statement created in release 11.5 or later was deleted, run **sp_checksource**. If the statement was deleted, you must either recreate the object that created the statement or reinstall the application that created the object, which will re-create the statement.

You can protect the text of a database object against unauthorized access by restricting **select** permission on the *text* column of the *syscomments* table to the owner of the object and the System Administrator. This restriction, which applies to direct access through **select** statements as well as access through stored procedures, is required in order to run Adaptive Server in the evaluated configuration. To enact this restriction, a System Security Officer must reset the parameter called **allow select on syscomments.text column** with the system procedure **sp_configure**. For information, see the *System Administration Guide*.

Indexes

Unique clustered index on *id, number, colid2, colid, texttype*

sysconfigures

(*master* database only)

Description

sysconfigures contains one row for each configuration parameter that can be set by the user.

Columns

Name	Datatype	Description
config	*smallint*	Configuration parameter number.
value	*int*	The user-modifiable value for the parameter with *integer* datatype. Its value is 0 for the parameters with *character* datatype.
comment	*varchar(255)*	Name of the configuration parameter.
status	*smallint*	Either 1 (dynamic) or 0 (parameter takes effect when Adaptive Server is restarted).
name	*varchar(80)*	Name of the configuration parameter (the same value as *comment*).
parent	*smallint*	Configuration parameter number of the parent; if more than one parent, the additional parent numbers are stored in *sysattributes*.
value2	*varchar(255)*	The user-modified value for the parameter with the character datatype. Its value is NULL for parameters with *integer* datatype. It is also used to store the pool size of a buffer pool.
value3	*int*	Stores the wash size of a buffer pool.
value4	*int*	Stores the asynchronous prefetch percents of a buffer pool.

Indexes

Unique clustered index on *name, parent, config*
Nonclustered index on *parent, config*
Nonclustered index on *config*

sysconstraints

(all databases)

Description

sysconstraints has one row for each referential constraint and check constraint associated with a table or column.

Whenever a user declares a new check constraint or referential constraint using **create table** or **alter table**, Adaptive Server inserts a row into the *sysconstraints* table. The row remains until a user executes **alter table** to drop the constraint. Dropping a table by executing **drop table** removes all rows associated with that table from the *sysconstraints* table.

Columns

Name	Datatype	Description
colid	*tinyint*	Column number in the table
spare1	*tinyint*	Unused
constrid	*int*	Object ID of the constraint
tableid	*int*	ID of the table on which the constraint is declared
error	*int*	Constraint specific error message
status	*int*	The type of constraint: 0x0040 = a referential constraint 0x0080 = a check constraint
spare2	*int*	Unused

Indexes

Clustered index on *tableid*, *colid*
Unique nonclustered index on *constrid*

syscoordinations

(*sybsystemdb* database only)

Description

syscoordinations contains information about remote Adaptive Servers participating in distributed transactions (remote participants) and their coordination states.

Columns

Name	Datatype	Description
participant	*smallint*	Participant ID
starttime	*datetime*	Date the transaction started
coordtype	*tinyint*	Value indicating the coordination method or protocol in the *systransactions* table definition
owner	*tinyint*	Row owner (for internal use)
protocol	*smallint*	Reserved for internal use
state	*smallint*	Value indicating the current state of the remote participant (see Table 11-4)
bootcount	*int*	Reserved for internal use
dbid	*smallint*	Database ID at the start of the transaction.
logvers	*tinyint*	Reserved for internal use
spare	*smallint*	Reserved for internal use
status	*tinyint*	Reserved for internal use
xactkey	*binary(14)*	Unique Adaptive Server transaction key
gtrid	*varchar(255)*	Global transaction ID for distributed transactions coordinated by Adaptive Server (reserved for internal use)
partdata	*varbinary(255)*	Reserved for internal use
srvname	*varchar(30)*	Name of local server (null for remote servers)

Table 11-4 lists the values for the *state* column:

Table 11-4: syscoordinations state values

state Value	Participant State
1	Begun
4	Prepared
7	Committed
9	In Abort Tran

Indexes

Unique clustered index on *xactkey, participant, owner*

syscurconfigs

(*master* database only)

Description

syscurconfigs is built dynamically when queried. It contains an entry for each of the configuration parameters, as does *sysconfigures*, but with the current values rather than the default values. In addition, it contains four rows that describe the configuration structure.

Columns

Name	Datatype	Description
config	*smallint*	Configuration parameter number.
value	*int*	The current run value for the parameter with *integer* datatype. Its value is 0 for the parameters with character datatype.
comment	*varchar(255)*	Amount of memory used by each configuration parameter, represented in a string format. Values marked with a hash mark (#) share memory with other parameters.
status	*smallint*	Either 1 (dynamic) or 0 (parameter takes effect when Adaptive Server is restarted).
value2	*varchar(255)*	The current run value for the parameter with the *character* datatype. Its value is NULL for parameters with the *integer* datatype.
defvalue	*varchar(255)*	Default value of the configuration parameter.
minimum_value	*int*	Minimum value of the configuration parameter.
maximum_value	*int*	Maximum value of the configuration parameter.
memory_used	*int*	Integer value for the amount of memory used by each configuration parameter.
display_level	*int*	Display level of the configuration parameter (the values are 1, 5, and 10).
datatype	*int*	Datatype of the configuration parameter.
message_num	*int*	Message number of the **sp_helpconfig** message for this configuration parameter.
apf_percent	*int*	The current run value for the asynchronous prefetch percent for a buffer pool. Valid only for rows that represent buffer pools.

sysdatabases

(*master* database only)

Description

sysdatabases contains one row for each database in Adaptive Server. When Adaptive Server is installed, *sysdatabases* contains entries for the *master* database, the *model* database, the *sybsystemprocs* database, and the *tempdb* database. If you have installed auditing, it also contains an entry for the *sybsecurity* database.

Columns

Name	Datatype	Description
name	*sysname*	Name of the database
dbid	*smallint*	Database ID
suid	*smallint*	Server user ID of database owner
status	*smallint*	Control bits; those that the user can set with **sp_dboption** are so indicated in Table 11-5
version	*smallint*	Unused
logptr	*int*	Pointer to transaction log
crdate	*datetime*	Creation date
dumptrdate	*datetime*	Date of the last **dump transaction**
status2	*intn*	Additional control bits (see Table 11-6)
audflags	*intn*	Audit settings for database
deftabaud	*intn*	Bit-mask that defines default audit settings for tables
defvwaud	*intn*	Bit-mask that defines default audit settings for views
defpraud	*intn*	Bit-mask that defines default audit settings for stored procedures
def_remote_type	*smallint*	Identifies the default object type to be used for remote tables if no storage location is provided via the stored procedure **sp_addobjectdef**

Name	Datatype	Description
def_remote_loc	*varchar(255)*	Identifies the default storage location to be used for remote tables if no storage location is provided via the stored procedure **sp_addobjectdef**
status3	*intn*	Additional control bits.
status4	*intn*	Additional control bits.

Table 11-5 lists the bit representations for the *status* column.

Table 11-5: status control bits in the sysdatabases table

Decimal	Hex	Status
4	0x04	**select into/bulkcopy**; can be set by user
8	0x08	**trunc log on chkpt**; can be set by user
16	0x10	**no chkpt on recovery**; can be set by user
32	0x20	Database created with **for load** option, or crashed while loading database, instructs recovery not to proceed
256	0x100	Database suspect; not recovered; cannot be opened or used; can be dropped only with **dbcc dbrepair**
512	0x200	**ddl in tran**; can be set by user
1024	0x400	**read only**; can be set by user
2048	0x800	**dbo use only**; can be set by user
4096	0x1000	**single user**; can be set by user
8192	0x2000	**allow nulls by default**; can be set by user

Table 11-6 lists the bit representations for the *status2* column.

Table 11-6: status2 control bits in the sysdatabases table

Decimal	Hex	Status
1	0x0001	**abort tran on log full**; can be set by user
2	0x0002	**no free space acctg**; can be set by user
4	0x0004	**auto identity**; can be set by user
8	0x0008	**identity in nonunique index**; can be set by user
16	0x0010	Database is offline
32	0x0020	Database is offline until recovery completes
64	0x0040	Database is being recovered (internal use)
128	0x0080	Database has suspect pages
512	0x0200	Database is in the process of being upgraded
1024	0x0400	Database brought online for standby access
-32768	0xFFFF8000	Database has some portion of the log which is not on a log-only device

Indexes

Unique clustered index on *name*
Unique nonclustered index on *dbid*

sysdepends

(all databases)

Description

sysdepends contains one row for each procedure, view, or table that is referenced by a procedure, view, or trigger.

Columns

Name	Datatype	Description
id	*int*	Object ID
number	*smallint*	Procedure number
depid	*int*	Dependent object ID
depnumber	*smallint*	Dependent procedure number
status	*smallint*	Internal status information
selall	*bit*	On if object is used in **select** * statement
resultobj	*bit*	On if object is being updated
readobj	*bit*	On if object is being read

Indexes

Unique clustered index on *id, number, depid, depnumber*

sysdevices

(*master* database only)

Description

sysdevices contains one row for each tape dump device, disk dump device, disk for databases, and disk partition for databases. On the Adaptive Server distribution media, there are four entries in *sysdevices*: one for the master device (for databases), one for a disk dump device, and two for tape dump devices.

Columns

Name	Datatype	Description
low	*int*	First virtual page number on database device (not used for dump devices)
high	*int*	Last virtual page number on database device or dump device
status	*smallint*	Bitmap indicating type of device, default and mirror status (see Table 11-7)
cntrltype	*smallint*	Controller type (0 if database device, 2 if disk dump device or streaming tape, 3–8 if tape dump device)
name	*sysname*	Logical name of dump device or database device
phyname	*varchar(127)*	Name of physical device
mirrorname	*varchar(127)*	Name of mirror device

The bit representations for the *status* column, shown in Table 11-7, are additive. For example, “3” indicates a physical disk that is also a default.

Table 11-7: status control bits in the sysdevices table

Decimal	Hex	Status
1	0x01	Default disk
2	0x02	Physical disk
4	0x04	Logical disk (not used)
8	0x08	Skip header
16	0x10	Dump device
32	0x20	Serial writes
64	0x40	Device mirrored

Table 11-7: status control bits in the sysdevices table (continued)

Decimal	Hex	Status
128	0x80	Reads mirrored
256	0x100	Secondary mirror side only
512	0x200	Mirror enabled
1024	0x400	Master device is mirrored
2048	0x800	Mirror disabled (used internally)
4096	0x1000	Primary device needs to be unmirrored (used internally)
8192	0x2000	Secondary device needs to be unmirrored (used internally)
16384	0x4000	UNIX file device uses **dsync** setting (writes occur directly to physical media)

Indexes

Unique clustered index on *name*

sysengines

(*master* database only)

Description

sysengines contains one row for each Adaptive Server engine currently online.

Columns

Name	Datatype	Description
engine	*smallint*	Engine number
osprocid	*int*	Operating system process ID (may be NULL)
osprocname	*char*	Operating system process name (may be NULL)
status	*char*	One of: online, in offline, in create, in destroy, debug, bad status
affinitied	*int*	Number of Adaptive Server processes with affinity to this engine
cur_kpid	*int*	Kernel process ID of process currently running on this engine, if any
last_kpid	*int*	Kernel process ID of process that previously ran on this engine
idle_1	*tinyint*	Reserved
idle_2	*tinyint*	Reserved
idle_3	*tinyint*	Reserved
idle_4	*tinyint*	Reserved
starttime	*datetime*	Date and time engine came online

sysgams

(all databases)

Description

sysgams stores the global allocation map (GAM) for the database. The GAM stores a bitmap for all allocation units of a database, with one bit per allocation unit. You cannot select from or view *sysgams*.

sysindexes

(all databases)

Description

sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains *text* or *image* columns.

Name	Datatype	Description
name	*sysname*	Index or table name
id	*int*	ID of a table, or ID of table to which index belongs
indid	*smallint*	0 if a table; 1 if a clustered index on an allpages-locked table; >1 if a nonclustered index or a clustered index on a data-only-locked table; 255 if *text*, *image* or Java off-row structure (LOB structure)
doampg	*int*	Page number for the object allocation map of a table
ioampg	*int*	Page number for the allocation map of an index or (LOB structure)
oampgtrips	*int*	Number of times OAM pages cycle in the cache without being re-used, before being flushed
status2	*int*	Internal system status information (see Table 11-8)
ipgtrips	*int*	Number of times index pages cycle in the cache, without being reused, before being flushed
first	*int*	If *indid* is 0 or 1, page number of the first data page. If *indid* is between 2 and 250, page number of first leaf-level index page.
root	*int*	If *indid* is 0 and table is an unpartitioned allpages-locked table, page number of last page of page chain; unused for other types of pages. If *indid* is between 1 and 250, page number of root of index tree.
distribution	*int*	Unused. Formerly used to store the page number of the distribution page for an index.

Name	Datatype	Description
usagecnt	*smallint*	Reserved
segment	*smallint*	Number of segment in which object resides
status	*smallint*	Internal system status information (see Table 11-9)
maxrowsperpage	*smallint*	Maximum number of rows per page
minlen	*smallint*	Minimum size of a row
maxlen	*smallint*	Maximum size of a row
maxirow	*smallint*	Maximum size of a non-leaf index row
keycnt	*smallint*	Number of keys for a clustered index on an allpages-locked table; number of keys, plus 1 for all other indexes
keys1	*varbinary(255)*	Description of key columns if entry is an index
keys2	*varbinary(255)*	Description of key columns if entry is an index
soid	*tinyint*	Sort order ID that the index was created with; 0 if there is no character data in the keys
csid	*tinyint*	Character set ID that the index was created with; 0 if there is no character data in the keys
base_partition	*int*	Partition number, incremented by **alter table...unpartition** commands
fill_factor	*smallint*	Fillfactor set for an index
res_page_gap	*smallint*	Value for the **reservepagegap** on a table
exp_rowsize	*smallint*	Expected size of data rows
keys3	*varbinary(255)*	Description of key columns if entry is an index
identitygap	*intn*	Identity gap for a table

Table 11-8 lists the bit representations for the *status2* column.

Table 11-8: status2 bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Index supports foreign key constraint
2	0x2	Index supports primary key / unique declarative constraint
4	0x4	Index includes an IDENTITY column
8	0x8	Constraint name not specified
16	0x10	Large I/Os (prefetch) not enabled for table, index, or text chain
32	0x20	MRU cache strategy not enabled for table, index, or text chain
64	0x40	Ascending inserts turned on for the table
256	0x0100	Index is presorted and does not need to be copied to new extents
512	0x0200	Table is a data-only-locked table with a clustered index
8192	0x2000	Index on a data-only-locked table is suspect

Table 11-9 lists the bit representations for the *status* column.

Table 11-9: status bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Abort current command or trigger if attempt to insert duplicate key
2	0x2	Unique index
4	0x4	Abort current command or trigger if attempt to insert duplicate row; always 0 for data-only-locked tables
16	0x10	Clustered index
64	0x40	Index allows duplicate rows, if an allpages-locked table; always 0 for data-only-locked tables
128	0x80	Sorted object; not set for tables without clustered indexes or for text objects
512	0x200	**sorted data** option used in **create index** statement
2048	0x800	Index on primary key
32768	0x8000	Suspect index; index was created under another sort order

Indexes

Unique clustered index on *id, indid*

sysjars

(all databases)

sysjars contains one row for each Java archive (JAR) file that is retained in the database. Uses row-level locking.

For more information about JAR files, Java classes, and Java datatypes, see *Java in Adaptive Server Enterprise*.

Columns

Name	Datatype	Description
sensitivity	*sensitivity*	Used by the Secure Adaptive Server.
jid	*int*	The ID of the JAR.
jstatus	*int*	Internal status information. Unused.
jname	*varchar(255)*	The JAR name.
jbinary	*image*	The contents of the JAR: the Java classes.

Indexes

Unique placement index on *jid*
Unique non-clustered index on *jname*

syskeys

(all databases)

Description

syskeys contains one row for each primary, foreign, or common key.

Columns

Name	Datatype	Description
id	*int*	Object ID
type	*smallint*	Record type
depid	*int null*	Dependent object ID
keycnt	*int null*	Number of non-null keys
size	*int null*	Reserved
key1	*int null*	Column ID
key2	*int null*	Column ID
key3	*int null*	Column ID
key4	*int null*	Column ID
key5	*int null*	Column ID
key6	*int null*	Column ID
key7	*int null*	Column ID
key8	*int null*	Column ID
depkey1	*int null*	Column ID
depkey2	*int null*	Column ID
depkey3	*int null*	Column ID
depkey4	*int null*	Column ID
depkey5	*int null*	Column ID
depkey6	*int null*	Column ID
depkey7	*int null*	Column ID
depkey8	*int null*	Column ID
spare1	*smallint*	Reserved

Indexes

Clustered index on *id*

syslanguages

(*master* database only)

Description

syslanguages contains one row for each language known to Adaptive Server. us_english is not in *syslanguages*, but it is always available to Adaptive Server.

Columns

Name	Datatype	Description
langid	*smallint*	Unique language ID
dateformat	*char(3)*	Date order; for example, “dmy”
datefirst	*tinyint*	First day of the week—1 for Monday, 2 for Tuesday, and so on, up to 7 for Sunday
upgrade	*int*	Adaptive Server version of last upgrade for this language
name	*varchar(30)*	Official language name, for example, “french”
alias	*varchar(30)*	Alternate language name, for example, “français”
months	*varchar(251)*	Comma-separated list of full-length month names, in order from January to December—each name is at most 20 characters long
shortmonths	*varchar(119)*	Comma-separated list of shortened month names, in order from January to December—each name is at most 9 characters long
days	*varchar(216)*	Comma-separated list of day names, in order from Monday to Sunday—each name is at most 30 characters long

Indexes

Unique clustered index on *langid*
Unique nonclustered index on *name*
Unique nonclustered index on *alias*

syslisteners

(*master* database only)

Description

syslisteners contains a row for each network protocol available for connecting with the current Adaptive Server. Adaptive Server builds *syslisteners* dynamically when a user or client application queries the table.

Columns

Name	Datatype	Description
net_type	*char(32)*	Network protocol
address_info	*char(255)*	Information that uniquely identifies this Adaptive Server on the network, usually the name of the current Adaptive Server and an identifying number, such as the server's port number for the protocol

syslocks

(*master* database only)

Description

syslocks contains information about active locks. It is built dynamically when queried by a user. No updates to *syslocks* are allowed.

Name	Datatype	Description
id	*int*	Table ID
dbid	*smallint*	Database ID
page	*int*	Page number
type	*smallint*	Type of lock (bit values for the *type* column are listed in Table 11-10)
spid	*smallint*	ID of process that holds the lock
class	*char(30)*	Name of the cursor this lock is associated with, if any
fid	*smallint*	The family (coordinating process and its worker processes) to which the lock belongs. *fid* values are listed in Table 11-11.
context	*tinyint*	Context type of lock request. *context* values are listed in Table 11-12.
row	*smallint*	Row number
loid	*int*	Unique lock owner ID

Table 11-10 lists the bit representations for the *type* column.

Table 11-10:type control bits in the syslocks table

Decimal	Hex	Status
1	0x1	Exclusive table lock
2	0x2	Shared table lock
3	0x3	Exclusive intent lock
4	0x4	Shared intent lock
5	0x5	Exclusive page lock
6	0x6	Shared page lock
7	0x7	Update page lock
8	0x8	Exclusive row lock
9	0x9	Shared row lock

Table 11-10:type control bits in the syslocks table (continued)

Decimal	Hex	Status
10	0xA	Update row lock
11	0xB	Shared next key lock
256	0x100	Lock is blocking another process
512	0x200	Demand lock

Table 11-11 lists the values for the *fid* column:

Table 11-11:fid column values in the syslocks table

Value	Interpretation
0	The task represented by the *spid* is a single task executing a statement in serial.
Nonzero	The task (*spid*) holding the lock is a member of a family executing a statement in parallel. If the value is equal to the *spid*, it indicates that the task is the coordinating process in a family executing a query in parallel.

Table 11-12 lists the values for the *context* column:

Table 11-12:context column values in the syslocks table

Value	Interpretation
null	The task holding this lock is either executing a query in serial, or it is a query being executed in parallel in transaction isolation level 1.
0x1	The task holding the lock will hold the lock until the query is complete. A lock's context may be "Fam dur" when: • The lock is a table lock held as part of a parallel query. • The lock is held by a worker process at transaction isolation level 3. • The lock is held by a worker process in a parallel query and must be held for the duration of the transaction.
0x2	Range lock held by serializable read task
0x4	Infinity key lock
0x8	Lock acquired on an index pages of an allpages-locked table
0x10	Lock on a page or row acquired to delete a row
0x20	Address lock acquired on an index page during a shrink or split operation
0x40	Intent lock held by a transaction performing repeatable reads. Valid only for shared intent and exclusive intent locks on data-only locked tables.

sysloginroles

***(master* database only)**

Description

sysloginroles contains a row for each instance of a server login possessing a system role. One row is added for each role granted to each login. For example, if a single server user is granted **sa_role**, **sso_role**, and **oper_role**, three rows are added to *sysloginroles* associated with that user's system user ID (*suid).*

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID
srid	*smallint*	Server role ID; one of the following: 0 = **sa_role** 1 = **sso_role** 2 = **oper_role** 4 = **navigator_role** 5 = **replication_role**
status	*smallint*	Reserved

Indexes

Clustered index on *suid*

syslogins

(*master* database only)

Description

syslogins contains one row for each valid Adaptive Server user account.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID
status	*smallint*	Status of the account (see Table 11-13)
accdate	*datetime*	Date *totcpu* and *totio* were last cleared
totcpu	*int*	CPU time accumulated by login
totio	*int*	I/O accumulated by login
spacelimit	*int*	Reserved
timelimit	*int*	Reserved
resultlimit	*int*	Reserved
dbname	*sysname*	Name of database in which to put user when connection established
name	*sysname*	Login name of user
password	*varbinary*	Password of user (encrypted)
language	*varchar(30)*	User's default language
pwdate	*datetime*	Date the password was last changed
audflags	*int*	User's audit settings
fullname	*varchar(30)*	Full name of the user
srvname	*varchar*(30)	Name of server to which a passthrough connection must be established if the AUTOCONNECT flag is turned on.
logincount	*smallint*	Number of failed login attempts; reset to 0 by a successful login.

On the Adaptive Server distribution media, *syslogins* contains an entry in which the name is "sa", the *suid* is 1, and the password is null. It also contains the entry "probe" with an unpublished password. The login "probe" and the user "probe" exist for the two

phase commit probe process, which uses a challenge and response mechanism to access Adaptive Server.

Table 11-13 lists the bit representations for the *status* column:

Table 11-13:status control bits in the syslogins table

Decimal	Hex	Status
1	0x1	Password contains fewer than 6 characters or is NULL
2	0x2	Account is locked
4	0x4	Password has expired

Indexes

Unique clustered index on *suid*
Unique nonclustered index on *name*

syslogs

(all databases)

Description

syslogs contains the transaction log. It is used by Adaptive Server for recovery and roll forward. It is not useful to users.

You cannot delete from, insert into, or update *syslogs*. Every data modification operation is logged, so before you can change *syslogs*, the change must be logged. This means that a change operation on *syslogs* adds a row to *syslogs*, which then must be logged, adding another row to *syslogs*, and so on, producing an infinite loop. The loop continues until the database becomes full.

Columns

Name	Datatype	Description
xactid	*binary(6)*	Transaction ID
op	*tinyint*	Number of update operation

syslogshold

(*master* database only)

Description

syslogshold contains information about each database's oldest active transaction (if any) and the Replication Server truncation point (if any) for the transaction log, but it is not a normal table. Rather, it is built dynamically when queried by a user. No updates to *syslogshold* are allowed.

Columns

Name	Datatype	Description
dbid	*smallint*	Database ID.
reserved	*int*	Unused.
spid	*smallint*	Server process ID of the user that owns the oldest active transaction (always 0 for Replication Server).
page	*int*	Starting page number of active portion in *syslogs* defined by oldest transaction (or the truncation page in *syslogs* for Replication Server).
xactid	*char(6)*	ID of the oldest active transaction (always 0x000000 for Replication Server).
masterxactid	*char(6)*	ID of the transaction's master transaction (if any) for multi-database transactions; otherwise 0x000000 (always 0x000000 for Replication Server).
starttime	*datetime*	Date and time the transaction started (or when the truncation point was set for Replication Server).

Name	Datatype	Description
name	*char(67)*	Name of the oldest active transaction. It is the name defined with **begin transaction**, "$user_transaction" if no value is specified with **begin transaction**, or "$chained_transaction" for implicit transactions started by the ANSI chained mode. Internal transactions started by Adaptive Server have names that begin with the dollar sign ($) and are named for the operation, or are named "$replication_truncation_point" for Replication Server.

sysmessages

(*master* database only)

Description

sysmessages contains one row for each system error or warning that can be returned by Adaptive Server. Adaptive Server displays the error description on the user's screen.

Columns

Name	Datatype	Description
error	*int*	Unique error number
severity	*smallint*	Severity level of error
dlevel	*smallint*	Reserved
description	*varchar(255)*	Explanation of error with placeholders for parameters
langid	*smallint*	Language; null for us_english
sqlstate	*varchar(5)*	SQLSTATE value for the error

Indexes

Clustered index on *error, dlevel*
Unique nonclustered index on *error, dlevel, langid*

sysmonitors

(*master* database only)

Description

sysmonitors contains one row for each monitor counter.

Columns

Name	Datatype	Description
field_name	*char(79)*	Name of the counter
group_name	*char(25)*	Group this counter belongs to
field_id	*smallint*	Unique identifier for the row
value	*int*	Current value of the counter
description	*char(255)*	Description of the counter; not used

sysobjects

(all databases)

Description

sysobjects contains one row for each table, view, stored procedure, extended stored procedure, log, rule, default, trigger, check constraint, referential constraint, and (in *tempdb* only) temporary object.

Columns

Name	Datatype	Description
name	*sysname*	Object name
id	*int*	Object ID
uid	*smallint*	User ID of object owner
type	*char(2)*	One of the following object types: D = default L = log P = procedure PR = prepare objects (created by Dynamic SQL) R = rule RI = referential constraint S = system table TR = trigger U = user table V = view XP = extended stored procedure
userstat	*smallint*	Application-dependent type information (32768 decimal [0x8000 hex] indicates to Data Workbench® that a procedure is a report)
sysstat	*smallint*	Internal status information (256 decimal [0x100 hex] indicates that table is read-only)
indexdel	*smallint*	Index delete count (incremented if an index is deleted)
schemacnt	*smallint*	Count of changes in the schema of an object (incremented if a rule or default is added)
sysstat2	*int*	Additional internal status information (see Table 11-14)
crdate	*datetime*	Date the object was created
expdate	*datetime*	Reserved

Name	Datatype	Description
deltrig	*int*	Stored procedure ID of a delete trigger if the entry is a table. Table ID if the entry is a trigger.
instrig	*int*	Stored procedure ID of a table's insert trigger if the entry is a table
updtrig	*int*	Stored procedure ID of a table's update trigger if the entry is a table
seltrig	*int*	Reserved
ckfirst	*int*	ID of first check constraint on the table
cache	*smallint*	Reserved
audflags	*int*	Object's audit settings
objspare	*int*	Spare
versionts	*binary*	

Table 11-14 lists the bit representations for the *sysstat2* column:

Table 11-14:sysstat2 control bits in the sysobjects table

Decimal	Hex	Status
1	0x1	Table has a referential constraint
2	0x2	Table has a foreign key constraint
4	0x4	Table has more than one check constraint
8	0x8	Table has a primary key constraint
16	0x10	Stored procedure can execute only in chained transaction mode
32	0x20	Stored procedure can execute in any transaction mode
64	0x40	Table has an IDENTITY field
512	0x200	Table does not contain variable-length columns
1024	0x400	Table is remote
2048	0x800	Table is a proxy table created with the **existing** keyword
8192	0x2000	Table uses allpages locking scheme
16384	0x4000	Table uses datapages locking scheme
32768	0x8000	Table uses datarows locking scheme
65536	0x10000	Table was created in a version 11.9 or later version of the server
131072	0x20000	Table has a clustered index
242144	0x40000	Object represents an Embedded SQL procedure

Indexes

Unique clustered index on *id*
Unique nonclustered index on *name, uid*

syspartitions

(all databases)

Description

syspartitions contains one row for each partition (page chain) of a partitioned table.

Columns

Name	Datatype	Description
state	*smallint*	Internal information about the state of the partition
id	*int*	Object ID of the partitioned table
partitionid	*int*	Partition ID number
firstpage	*int*	Page number of the partition's first page
controlpage	*int*	Page number of the partition's control page
spare	*binary(32)*	Reserved

Indexes

Unique clustered index on *id, partitionid*

sysprocedures

(all databases)

Description

sysprocedures contains entries for each view, default, rule, trigger, procedure, declarative default, and check constraint. The plan or sequence tree for each object is stored in binary form. If the sequence tree does not fit into one entry, it is broken into more than one row. The *sequence* column identifies the sub-rows.

Columns

Name	Datatype	Description
type	*smallint*	Object type (see Table 11-15)
id	*int*	Object ID
sequence	*smallint*	Sequence number if more than one row is used to describe this object
status	*smallint*	Internal system status
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
version	*int*	

Table 11-15 lists the bit representations for the *type* column.

Table 11-15:type control bits in the sysprocedures table

Decimal	Hex	Status
1	0x1	Entry describes a plan (reserved)
2	0x2	Entry describes a tree

Indexes

Unique clustered index on *id, type, sequence, number*

sysprocesses

(*master* database only)

Description

sysprocesses contains information about Adaptive Server processes, but it is not a normal table. Rather, it is built dynamically when queried by a user. No updates to *sysprocesses* are allowed.

Use the **kill** statement to kill a process.

Columns

Name	Datatype	Description
spid	*smallint*	Process ID
kpid	*int*	Kernel process ID
enginenum	*int*	Number of engine on which process is being executed
status	*char(12)*	Process ID status (see Table 11-16)
suid	*smallint*	Server user ID of user who issued command
hostname	*char(10)*	Name of host computer
program_name	*char(16)*	Name of front-end module
hostprocess	*char(8)*	Host process ID number
cmd	*char(16)*	Command currently being executed
cpu	*int*	Cumulative CPU time for process in ticks
physical_io	*int*	Number of disk reads and writes for current command
memusage	*int*	Amount of memory allocated to process
blocked	*smallint*	Process ID of blocking process, if any
dbid	*smallint*	Database ID
uid	*smallint*	ID of user who executed command
gid	*smallint*	Group ID of user who executed command
tran_name	*varchar(64)*	Name of the active transaction
time_blocked	*int*	Time blocked in seconds
network_pktsz	*int*	Current connection's network packet size
fid	*smallint*	Process ID of the worker process' parent
execlass	*varchar(30)*	Execution class that the process is bound to

Name	Datatype	Description
priority	*varchar(10)*	Base priority associated with the process
affinity	*varchar(30)*	Name of the engine to which the process has affinity
id	*int*	Object ID of the currently running procedure (or 0 if no procedure is running)
stmtnum	*int*	The current statement number within the running procedure (or the SQL batch statement number if no procedure is running)
linenum	*int*	The line number of the current statement within the running stored procedure (or the line number of the current SQL batch statement if no procedure is running)
origsuid	*smallint*	Original server user ID. If this value is not NULL, a user with an *suid* of *origsuid* executed **set proxy** or **set session authorization** to impersonate the user who executed the command.
block_xloid	*int*	Unique lock owner ID of a lock that is blocking a transaction
clientname	*varchar(30)*	Name by which the user is know for the current session. This parameter is optional
clienthostname	*varchar(30)*	Name by which the host is known for the current session. This parameter is optional
clientapplname	*varchar(30)*	Name by which the application is known for the current session. This parameter is optional
sys_id	*smallint*	Unique identity of companion node
ses_id	*int*	Unique identity of each client session

Table 11-16 lists the values for the *status* column:

Table 11-16:sysprocesses status column values

Status	Meaning
alarm sleep	Waiting for alarm to wake process up (user executed a **waitfor delay** command)
background	A process, such as a threshold procedure, run by Adaptive Server rather than by a user process
infected	Server has detected a serious error condition; extremely rare
latch sleep	Waiting on a latch acquisition

Table 11-16:sysprocesses status column values

Status	Meaning
lock sleep	Waiting on a lock acquisition
log suspend	Processes suspended by reaching the last-chance threshold on the log
PLC sleep	Waiting to access a user log cache
recv sleep	Waiting on a network read
runnable	In the queue of runnable processes
running	Actively running on one of the server engines
send sleep	Waiting on a network send
sleeping	Waiting on a disk I/O, or some other resource (often indicates a process that is running, but doing extensive disk I/O)
stopped	Stopped process
sync sleep	Waiting on a synchronization message from another process in the family

sysprotects

(all databases)

Description

sysprotects contains information on permissions that have been granted to, or revoked from, users, groups, and roles.

Columns

Name	Datatype	Description
id	*int*	ID of the object to which this permission applies.
uid	*smallint*	ID of the user, group, or role to which this permission applies.
action	*tinyint*	One of the following permissions: 167 = **set proxy** or **set session authorization** 193 = **select** 195 = **insert** 196 = **delete** 197 = **update** 224 = **execute** 151 = **references** 203 = **create database** 233 = **create default** 222 = **create procedure** 236 = **create rule** 198 = **create table** 207 = **create view** 228 = **dump database** 235 = **dump transaction**
protecttype	*tinyint*	One of the following values: 0 = **grant with grant** 1 = **grant** 2 = **revoke**
columns	*varbinary(32)*	Bitmap of columns to which this **select** or **update** permission applies. Bit 0 indicates all columns; 1 means permission applies to that column; NULL means no information.
grantor	*smallint*	User ID of the grantor (or of object owner if grantor is a System Administrator).

Indexes

Unique clustered index on *id, action, grantor, uid, protecttype*

sysqueryplans

(all databases)

Description

sysqueryplans contains two or more rows for each abstract query plan. Uses datarow locking.

Columns

Name	Datatype	Description
uid	*smallint*	User ID of user who captured the abstract plan.
gid	*int*	The abstract plan group ID under which the abstract plan was saved.
hashkey	*int*	The hash key over the SQL query text.
id	*int*	The unique ID if the abstract plan.
type	*smallint*	10 if the text column contains query text or 100 if the text column contains abstract plan text.
sequence	*smallint*	Sequence number if multiple rows are required for the text of the SQL query or abstract plan.
status	*int*	Reserved.
text	*varchar(255)*	The SQL text, if *type* is 10, or the abstract query plan text, if the *type* is 100.

Indexes

Unique clustered index on *uid, gid, hashkey, id, type, sequence*
Nonclustered unique index on *id, type, sequence*

sysreferences

(all databases)

Description

sysreferences contains one row for each referential integrity constraint declared on a table or column.

Columns

Name	Datatype	Description
indexid	*smallint*	ID of the unique index on referenced columns
constrid	*int*	Object ID of the constraint from *sysobjects*
tableid	*int*	Object ID of the referencing table
reftabid	*int*	Object ID of the referenced table
keycnt	*tinyint*	Number of columns in the foreign key
status	*smallint*	Reserved
frgndbid	*smallint*	Database ID of the database that includes the referenced table (the table with the foreign key).
pmrydbid	*smallint*	Database ID of the database that includes the referenced table (the table with the primary key).
spare2	*int*	Reserved
fokey1 . . .	*tinyint*	Column ID of the first referencing column
fokey16	*tinyint*	Column ID of the sixteenth referencing column
refkey1 . . .	*tinyint*	Column ID of the first referenced column
refkey16	*tinyint*	Column ID of the sixteenth referenced column
frgndbname	*varchar(30)*	Name of the database that includes the referencing table (the table with the foreign key); NULL if the referencing table is in the current database
pmrydbname	*varchar(30)*	Name of the database that includes the referenced table (the table with the primary key); NULL if the referenced table is in the current database

Indexes

Clustered index on *tableid, frgndbname*
Nonclustered index on *constrid, frgndbname*
Nonclustered index on *reftabid, indexid, pmrydbname*

sysremotelogins

(*master* database only)

Description

sysremotelogins contains one row for each remote user that is allowed to execute remote procedure calls on this Adaptive Server.

Columns

Name	Datatype	Description
remoteserverid	*smallint*	Identifies the remote server
remoteusername	*varchar(30)*	User's login name on remote server
suid	*smallint*	Local server user ID
status	*smallint*	Bitmap of options

Indexes

Unique clustered index on *remoteserverid, remoteusername*

sysresourcelimits

(*master* database only)

Description

sysresourcelimits contains a row for each resource limit defined by Adaptive Server. Resource limits specify the maximum amount of server resources that can be used by a Adaptive Server login or an application to execute a query, query batch, or transaction.

Columns

Name	Datatype	Description
name	*varchar(30) null*	Login name
appname	*varchar(30) null*	Application name
rangeid	*smallint*	*id* column from *systimeranges*
limitid	*smallint*	*id* column from *spt_limit_types*
limitvalue	*int*	Value of limit
enforced	*tinyint*	Subset of the *enforced* column from *spt_limit_types*: 1 = prior to execution 2 = during execution 3 = both
actiontotake	*tinyint*	Action to take on a violation: 1 = issue warning 2 = abort query batch 3 = abort transaction 4 = kill session
scope	*tinyint*	Scope of user limit (a bitmap indicating one or more of the following): 1 = query 2 = query batch 4 = transaction
spare	*tinyint*	Reserved

Indexes

Clustered index on *name*, *appname*

sysroles

(all databases)

Description

sysroles maps server role IDs to local role IDs.

Columns

Name	Datatype	Description
id	*smallint*	Server role ID (*srid*)
lrid	*smallint*	Local role ID
type	*smallint*	Unused
status	*smallint*	Unused

When a database permission is granted to a role, if an entry for the role does not exist in *syssrvroles*, Adaptive Server adds an entry to *sysroles* map the local role ID (*lrid*) to the server-wide role ID (*srid)* in *syssrvroles*.

Indexes

Unique clustered index on *lrid*

syssecmechs

(*master* database only)

Description

syssecmechs contains information about the security services supported by each security mechanism that is available to Adaptive Server. Unlike other system tables, it is not created during installation. Instead, it is built dynamically when queried by a user.

Columns

Name	Datatype	Description
sec_mech_name	*varchar(30)*	Name of the security mechanism; for example, “NT LANMANAGER”
available_service	*varchar(30)*	Name of the security service supported by the security mechanism; for example, “unified login”

syssegments

(all databases)

Description

syssegments contains one row for each segment (named collection of disk pieces). In a newly created database, the entries are: segment 0 (*system*) for system tables; segment 2 *(logsegment*) for the transaction log; and segment 1 (*default*) for other objects.

Columns

Name	Datatype	Description
segment	*smallint*	Segment number
name	*sysname*	Segment name
status	*int null*	Indicates which segment is the default segment

sysservers

(*master* database only)

Description

sysservers contains one row for each remote Adaptive Server, Backup Server™, or Open Server™ on which this Adaptive Server can execute remote procedure calls.

Columns

Name	Datatype	Description
srvid	*smallint*	ID number (for local use only) of the remote server
srvstatus	*smallint*	Bitmap of options (see Table 11-17)
srvname	*varchar(30)*	Server name
srvnetname	*varchar(32)*	Interfaces file name for the server
srvclass	*smallint*	Server category defined by the class parameter of **sp_addserver**. See Table 11-18.
srvsecmech	*varchar(30)*	Security mechanism

Table 11-17 lists the bit representations for the *srvstatus* column:

Table 11-17:status control bits in the sysservers table

Decimal	Hex	Status
0	0x0	Timeouts are enabled
1	0x1	Timeouts are disabled
2	0x2	Network password encryption is enabled
4	0x4	Remote server is read only
8	0x8	Use rpc security model A

Table 11-18 lists the server categories for the *srvclass* column:

Table 11-18:Server categories in the sysservers table

srvclass	Server category
0	Local server (this server)
1	Another Adaptive Server or Component Integration Services server
3	Server coded to the DirectCONNECT specification
4	Server accessible by Net-Gateway or MDI Database Gateway
5	Server coded to the Generic Access Module specification

Indexes

Unique clustered index on *srvid*
Unique nonclustered index on *srvname*

syssessions

(*master* database only)

Description

syssessions is only used when Adaptive Server is configured for Sybase's Failover in a high availability system. *syssessions* contains one row for each client that connects to Adaptive Server with the failover property (for example, **isql -Q**). Clients that have an entry in *syssessions* during failover are moved to the secondary companion. Clients that do not have an entry in *syssessions* are dropped during failover. Clients that have an entry in *syssessions* during failback are moved to the primary companion. Clients that do not have an entry in *syssessions* during failback are dropped.

Columns

Name	Datatype	Description
sys_id	*smallint*	Unique identity of companion node
ses_id	*int*	Unique identity of each client session
state	*tinyint*	Describes whether the session is active or inactive
spare	*tinyint*	Reserved for future functionality
status	*smallint*	Reserved for future functionality
dbid	*smallint*	Reserved for future functionality
name	*varchar(30)*	Same as client's login name as specified in *syslogins*

syssrvroles

(*master* database only)

Description

syssrvroles contains a row for each system or user-defined role.

Columns

Name	Datatype	Description
srid	*smallint*	Server role ID
name	*varchar*(30)	Name of the role
password	*varinary*(30)	Password for the role (encrypted)
pwdate	*datetime*	Date the password was last changed
status	*smallint*	Bitmap for role status. See Figure 11-19
logincount	*smallint*	Number of failed login attempts; reset to 0 by a successful login.

Table 11-19 lists the bit representations for the *status* column:

Table 11-19:status control bits in the syssrvroles table

Decimal	Hex	Status
2	0x2	Role is locked
4	0x4	Role is expired

Indexes

Unique clustered index on *srid*

sysstatistics

(all databases)

Description

sysstatistics contains one or more rows for each indexed column on a user table. May also contain rows for unindexed column. Uses datarow locking.

Name	Datatype	Description
statid	*smallint*	Reserved
id	*int*	Object ID of table
sequence	*int*	Sequence number if multiple rows are required for this set of statistics
moddate	*datetime*	Date this row was last modified
formatid	*tinyint*	Type of statistics represented by this row
usedcount	*tinyint*	Number of fields *c0* to *c79* used in this row
colidarray	*varbinary(100)*	An ordered list of column IDs
c0...c79	*varbinary(255)*	Statistical data

Indexes

Unique clustered index on *id, statid, colidarray, formatid, sequence*

systabstats

(all databases)

Description

systabstats contains one row for each clustered index, one row for each nonclustered index, and one row for each table that has no clustered index. Uses datarow locking.

Name	Datatype	Description
indid	*smallint*	0 if a table; 1 if a clustered index on an allpages-locked table; >1 if a nonclustered index or a clustered index on a data-only-locked table; 255 if *text* or *image* object
id	*int*	ID of table to which index belongs
activestatid	*smallint*	Reserved
indexheight	*smallint*	Height of the index; maintained if *indid* is greater than 1
leafcnt	*int*	Number of leaf pages in the index; maintained if *indid* is greater than 1
pagecnt	*int*	Number of pages in the table or index
rowcnt	*float*	Number of rows in the table; maintained for *indid* of 0 or 1
forwrowcnt	*float*	Number of forwarded rows; maintained for *indid* of 0 or 1
delrowcnt	*float*	Number of deleted rows
dpagecrcnt	*float*	Number of extent I/Os that need to be performed to read the entire table
ipagecrcnt	*float*	Number of extent I/Os that need to be performed to read the entire leaf level of a nonclustered index
drowcrcnt	*float*	Number of page I/Os that need to be performed to read an entire table
oamapgcnt	*int*	Number of OAM pages for the table, plus the number of allocation pages that store information about the table
extent0pgcnt	*int*	Count of pages that are on the same extent as the allocation page
datarowsize	*float*	Average size of the data row

Name	Datatype	Description
leafrowsize	*float*	Average size of a leaf row for nonclustered indexes and clustered indexes data-only-locked tables
status	*int*	Internal system status information (see Table 11-20)
spare1	*int*	Reserved
spare2	*float*	Reserved
rslastoam	*int*	Last OAM page visited by a **reorg reclaim_space** or **reorg compact** command
rslastpage	*int*	Last data or leaf page visited by a **reorg reclaim_space** or **reorg compact** command
frlastoam	*int*	Last OAM page visited by the **reorg forwarded_rows** command
frlastpage	*int*	Last data page visited by the **reorg forwarded_rows** command
conopt_thld	*smallint*	Concurrency optimization threshold
spare3	*int*	Reserved
emptypgcnt	*int*	Number of empty pages in extents allocated to the table or index
spare4	*float*	Reserved

Table 11-20 lists the bit representations for the *status* column:

Table 11-20:status bits in the systabstats table

Decimal	Hex	Status
1	0x1	Statistics are the result of upgrade (not **update statistics**)

Indexes

Unique clustered index on *id, indid*

systhresholds

(all databases)

Description

systhresholds contains one row for each threshold defined for the database.

Columns

Name	Datatype	Description
segment	*smallint*	Segment number for which free space is being monitored.
free_space	*int*	Size of threshold, in 2K pages (4K for Status).
status	*smallint*	Bit 1 equals 1 for the logsegment's last-chance threshold, 0 for all other thresholds.
proc_name	*varchar(255)*	Name of the procedure that is executed when the number of unused pages on *segment* falls below *free_space.*
suid	*smallint*	The server user ID of the user who added the threshold or modified it most recently.
currauth	*varbinary(255)*	A bit mask that indicates which roles were active for *suid* at the time the threshold was added or most recently modified. When the threshold is crossed, *proc_name* executes with this set of roles, less any that have been deactivated since the threshold was added or last modified.

Indexes

Unique clustered index on *segment, free_space*

systimeranges

(*master* database only)

Description

systimeranges stores named time ranges, which are used by Adaptive Server to control when a resource limit is active.

Columns

Name	Datatype	Description
name	*varchar(30)*	Unique name of the time range.
id	*smallint*	Unique identifier for the time range. 1 represents the "at all times" limit.
startday	*tinyint*	Day of week (1–7) for the beginning of the range. Monday = 1, Sunday = 7.
endday	*tinyint*	Day of week (1–7) for the end of the range. Monday = 1, Sunday = 7.
starttime	*varchar(10)*	Time of day for the beginning of the range.
endtime	*varchar(10)*	Time of day for the end of the range.

Indexes

Clustered index on *id*

systransactions

(*master* database only)

Description

systransactions contains information about Adaptive Server transactions, but it is not a normal table. Portions of the table are built dynamically when queried by a user, while other portions are stored in the master database. Updates to the dynamically-built columns of *systransactions* are not allowed.

Columns

Name	Datatype	Description
xactkey	*binary(14)*	Unique Adaptive Server transaction key
starttime	*datetime*	Date the transaction started
failover	*int*	Value indicating the transaction failover state (see Table 11-21)
type	*int*	Value indicating the type of transaction (see Table 11-22)
coordinator	*int*	Value indicating the coordination method or protocol (see Table 11-23)
state	*int*	Value indicating the current state of the transaction (see Table 11-24)
connection	*int*	Value indicating the connection state (see Table 11-25)
status	*int*	Internal transaction status flag
status2	*int*	Additional internal transaction status flags.
spid	*smallint*	Server process ID, or 0 if the process is detached
masterdbid	*smallint*	Starting database of the transaction
loid	*int*	Lock owner ID
namelen	*smallint*	Length of “xactname” below
xactname	*varchar(255)*	Transaction name or *XID*
srvname	*varchar(30)*	Name of the remote server (null for local servers)

Table 11-21 lists the values for the *failover* column:

Table 11-21:systransactions failover column values

failover Value	Failover State
0	Resident Tx
1	Failed-over Tx
2	Tx by Failover-Conn

Table 11-22 lists the values for the *type* column:

Table 11-22:systransactions type column values

type Value	Transaction Type
1	Local
3	External
98	Remote
99	Dtx_State

Table 11-23 lists the values for the *coordinator* column:

Table 11-23:systransactions coordinator column values

coordinator Value	Coordination Method or Protocol
0	None
1	Syb2PC
2	ASTC
3	XA
4	DTC

Table 11-24 lists the values for the *state* column:

Table 11-24:systransactions state column values

state Value	Transaction State
1	Begun
2	Done Command
3	Done
4	Prepared
5	In Command
6	In Abort Cmd
7	Committed
8	In Post Commit
9	In Abort Tran
10	In Abort Savept
65537	Begun-Detached
65538	Done Cmd-Detached
65539	Done-Detached
65540	Prepared-Detached
65548	Heur Committed
65549	Heur Rolledback

Table 11-25 lists the values for the *connection* column:

Table 11-25:systransactions connection column values

connection Value	Connection State
1	Attached
2	Detached

systypes

(all databases)

Description

systypes contains one row for each system-supplied and user-defined datatype. Domains (defined by rules) and defaults are given, if they exist.

The rows that describe system-supplied datatypes cannot be altered.

Columns

Name	Datatype	Description
uid	*smallint*	User ID of datatype creator
usertype	*smallint*	User type ID
variable	*bit*	1 if datatype is variable length; 0 otherwise
allownulls	*bit*	Indicates whether nulls are allowed for this datatype
type	*tinyint*	Physical storage datatype
length	*tinyint*	Physical length of datatype
tdefault	*int*	ID of system procedure that generates default for this datatype
domain	*int*	ID of system procedure that contains integrity checks for this datatype
name	*sysname*	Datatype name
printfmt	*varchar(255)*	Reserved
prec	*tinyint*	Number of significant digits
scale	*tinyint*	Number of digits to the right of the decimal point
ident	*tinyint*	1 if column has the IDENTITY property, 0 if it does not
hierarchy	*tinyint*	Precedence of the datatype in mixed mode arithmetic

Table 11-26 lists each system-supplied datatype's *name*, *hierarchy*, *type* (not necessarily unique), and *usertype* (unique). The datatypes are

ordered by *hierarchy*. In mixed-mode arithmetic, the datatype with the lowest *hierarchy* takes precedence:

Table 11-26:Datatype names, hierarchy, types, and usertypes

Name	*hierarchy*	*type*	*usertype*
floatn	1	109	14
float	2	62	8
datetimn	3	111	15
datetime	4	61	12
real	5	59	23
numericn	6	108	28
numeric	7	63	10
decimaln	8	106	27
decimal	9	55	26
moneyn	10	110	17
money	11	60	11
smallmoney	12	122	21
smalldatetime	13	58	22
intn	14	38	13
int	15	56	7
smallint	16	52	6
tinyint	17	48	5
bit	18	50	16
varchar	19	39	2
sysname	19	39	18
nvarchar	19	39	25
char	20	47	1
nchar	20	47	24
varbinary	21	37	4
timestamp	21	37	80
binary	22	45	3
text	23	35	19
image	24	34	20

Indexes

Unique clustered index on *name*
Unique nonclustered index on *usertype*

sysusages

(*master* database only)

Description

sysusages contains one row for each **disk allocation piece** assigned to a database. Each database contains a specified number of database (logical) page numbers. Each disk piece includes the segments on the Adaptive Server distribution media, segments 0 and 1.

The **create database** command checks *sysdevices* and *sysusages* to find available disk allocation pieces. One or more contiguous disk allocation pieces are assigned to the database, and the mapping is recorded in *sysusages*.

Columns

Name	Datatype	Description
dbid	*smallint*	Database ID
segmap	*int*	Bitmap of possible segment assignments
lstart	*int*	First database (logical) page number
size	*int*	Number of contiguous database (logical) pages
vstart	*int*	Starting virtual page number
pad	*smallint*	Unused
unreservedpgs	*int*	Free space not part of an allocated extent

Indexes

Unique clustered index on *dbid, lstart*
Unique nonclustered index on *vstart*

sysusermessages

(all databases)

Description

sysusermessages contains one row for each user-defined message that can be returned by Adaptive Server.

Columns

Name	Datatype	Description
error	*int*	Unique error number. Must be 20,000 or higher.
uid	*smallint*	Server user ID (**suser_id**) of the message creator.
description	*varchar(255)*	User-defined message with optional placeholders for parameters.
langid	*smallint*	Language ID for this message; null for us_english.
dlevel	*smallint*	Stores the with_log bit, which is used to call the appropriate routine to log a message.

Indexes

Clustered index on *error*
Unique nonclustered index on *error, langid*

sysusers

(all databases)

Description

sysusers contains one row for each user allowed in the database, and one row for each group or role.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID, copied from *syslogins*.
uid	*smallint*	User ID, unique in this database, is used for granting and revoking permissions. User ID 1 is "dbo".
gid	*smallint*	Group ID to which this user belongs. If *uid* = *gid*, this entry defines a group. The group "public" has *suid* = -2; all other groups have *suid* = - *gid*.
name	*sysname*	User or group name, unique in this database.
environ	*varchar(255)*	Reserved.

On the Adaptive Server distribution media, *master..sysusers* contains some initial users: "dbo", whose *suid* is 1 and whose *uid* is 1; "guest", whose *suid* is -1 and whose *uid* is 2; and "public", whose *suid* is -2 and whose *uid* is 0. In addition, both system-defined and user-defined roles (**sa_role**, **sso_role**, *role_name*) is listed in *sysusers*.

The user "guest" provides a mechanism for giving users that are not explicitly listed in *sysusers* access to the database with a restricted set of permissions. The "guest" entry in *master* means that any user with an account on Adaptive Server (that is, with an entry in *syslogins*) can access *master*.

The user "public" refers to all users. The keyword **public** is used with the **grant** and **revoke** commands to signify that permission is being given to or taken away from all users.

Indexes

Unique clustered index on *suid*
Unique nonclustered index on *name*
Unique nonclustered index on *uid*

sysxtypes

(all databases)

Description

sysxtypes contains one row for each extended, Java-SQL datatype. Uses row-level locking.

Refer to *Java in Adaptive Server Enterprise* for more information about Java-SQL classes and datatypes.

Columns

Name	Datatype	Description
sensitivity	*sensitivity*	Used by the Secure Adaptive Server.
xtname	*varchar(255)*	The name of the extended type.
xtid	*int*	System-generated ID for the extended type.
xtstatus	*int*	Internal status information. Unused.
xtmetatype	*int*	Unused.
xtcontainer	*int*	The ID of the JAR file containing the class. Can be NULL.
xtsource	*text*	Source code for the extended type. Unused.
xtbinary	*image*	Object code for the extended type. For Java classes, it contains the class file.

Indexes

Unique placement index on *xtid*
Unique non-clustered index on *xtname*

syblicenseslog

(*master* database only)

Description

syblicenseslog contains one row for each update of the maximum number of licenses used in Adaptive Server per 24-hour period. *syblicenseslog* is updated every 24 hours. If Adaptive Server is shut down at any time, License Use Manager logs the number of licenses currently being used in *syblicenseslog* before the shutdown is complete. The 24 hour period restarts when you start Adaptive Server.

➤ ***Note***

syblicenseslog is not a system table. Its type is "U" and its object ID is greater than 100.

Columns

Name	Datatype	Description
status	*smallint*	Status of the maximum number of licenses used; one of the following: • 0 = number of licenses not exceeded • 1 = number of licenses is exceeded • -1 = housekeeper is unable to monitor number of licenses
logtime	*datetime*	Date and time the log was written
maxlicenses	*int*	Maximum number of licenses used during the 24-hour period

12 *dbccdb* Tables

In addition to the standard system tables included in all databases, the **dbcc** management database, *dbccdb*, contains seven tables that define inputs to and outputs from **dbcc checkstorage**. It also contains at least two workspaces. Topics include:

dbcc_config

The *dbcc_config* table describes the currently executing or last completed **dbcc checkstorage** operation. It defines:

- The location of resources dedicated to the **dbcc checkstorage** operation
- Resource usage limits for the **dbcc checkstorage** operation

Table 12-1: Columns in the dbcc_config table

Column Name	Datatype	Description
dbid	*smallint*	Matches the *dbid* from a row in *sysdatabases*.
type_code	*int*	Matches the *type_code* from a row in the *dbcc_types* table. Valid values are 1–9.
value	*int*	Specifies the value of the item identified by *type_code*. Can be null only if the value of *stringvalue* is not null.

Table 12-1: Columns in the dbcc_config table (continued)

Column Name	Datatype	Description
stringvalue	*varchar(255)*	Specifies the value of the item identified by *type_code*. Can be null only if the value of *value* is not null.

Primary key: combination of *dbid* and *type_code*

For information on initializing and updating *dbcc_config*, see the *System Administration Guide*.

dbcc_counters

The *dbcc_counters* table stores the results of the analysis performed by **dbcc checkstorage**. Counters are maintained for each database, table, index, partition, device, and invocation of **dbcc**.

Table 12-2: Columns in the dbcc_counters table

Column Name	Datatype	Description
dbid	*smallint*	Identifies the target database.
id	*int*	Identifies the table. The value is derived from *sysindexes* and *sysobjects*.
indid	*smallint*	Identifies the index. The value is derived from *sysindexes*.
partitionid	*smallint*	Identifies the defined object-page affinity. The value is derived from *sysindexes* and *syspartitions*.
devid	*smallint*	Identifies the disk device. The value is derived from *sysdevices*.
opid	*smallint*	Identifies the **dbcc** operation that was performed.
type_code	*int*	Matches the *type_code* column of a row in the *dbcc_types* table. Valid values are 5000 through 5019.
value	*real*	Matches the appropriate *type_name* for the given *type_code* as described in *dbcc_types*.

Primary key: combination of *dbid*, *id*, *indid*, *partitionid*, *devid*, *opid*, and *type_code*

dbcc_fault_params

The *dbcc_fault_params* table provides additional descriptive information for a fault entered in the *dbcc_faults* table.

Table 12-3: Columns in the dbcc_fault_params table

Column Name	Datatype	Description
dbid	*smallint*	Identifies the target database.
opid	*smallint*	Identifies the **dbcc** operation that was performed.
faultid	*int*	Identifies the fault ID.
type_code	*int*	Defines the interpretation of the value, which is provided by the "value" columns. Valid values are 1000–1009. They are described in the *dbcc_types* table.
intvalue	*int*	Specifies the integer value.
realvalue	*real*	Specifies the real value.
binaryvalue	*varbinary(255)*	Specifies the binary value.
stringvalue	*varchar(255)*	Specifies the string value.
datevalue	*datetime*	Specifies the date value.

Primary key: combination of *dbid*, *opid*, *faultid*, and *type_code*

Each "value" column (*intvalue*, *realvalue*, *binaryvalue*, *stringvalue*, and *datevalue*) can contain a null value. At least one must not be null. If more than one of these columns contains a value other than null, the columns provide different representations of the same value.

dbcc_faults

The *dbcc_faults* table provides a description of each fault detected by **dbcc checkstorage**.

Table 12-4: Columns in the dbcc_faults table

Column Name	Datatype	Description
dbid	*smallint*	Identifies the target database.
id	*smallint*	Identifies the table. The value is derived from *sysindexes* and *sysobjects*.

Table 12-4: Columns in the dbcc_faults table (continued)

Column Name	Datatype	Description
indid	*smallint*	Identifies the index. The value is derived from *sysindexes*.
partitionid	*smallint*	Identifies the partition. The value is derived from *sysindexes* and *syspartitions*. Counters are maintained for page ranges, so "partition" refers to the defined object-page affinity, rather than the actual object page chain.
devid	*smallint*	Identifies the disk device. The value is derived from *sysdevices*
opid	*smallint*	Identifies the **dbcc** operation that was performed.
faultid	*int*	Provides a unique sequence number assigned to each fault recorded for the operation.
type_code	*int*	Identifies the type of fault. Valid values are 100000–100032. They are described in Table 12-7.
status	*int*	Classifies the fault. Valid values are: 0 = Soft fault, possibly spurious 1 = Hard fault For more information, see the *System Administration Guide*.

Primary key: combination of *dbid*, *id*, *indid*, *partitionid*, *devid*, *opid*, *faultid*, and *type_code*

dbcc_operation_log

The *dbcc_operation_log* table records the use of the **dbcc checkstorage** operations.

Table 12-5: Columns in the dbcc_operation_log table

Column Name	Datatype	Description
dbid	*smallint*	Identifies the target database

Table 12-5: Columns in the dbcc_operation_log table (continued)

Column Name	Datatype	Description
opid	*smallint*	Identifies the sequence number of the **dbcc checkstorage** operation. *opid* is an automatically incrementing number, unique for each *dbid* and *finish* pair.
optype	*smallint*	The following value is valid for *optype*: 2 = checkstorage
suid	*int*	Identifies the user executing the command
start	*datetime*	Identifies when the operation started
finish	*datetime*	Identifies when the operation ended

Primary key: combination of *dbid*, *opid*, and *optype*

Summary results are recorded in the *dbcc_operation_results* table.

dbcc_operation_results

The *dbcc_operation_results* table provides additional descriptive information for an operation recorded in the *dbcc_operation_log* table.

Table 12-6: Columns in the dbcc_operation_results table

Column Name	Datatype	Description
dbid	*smallint*	Identifies the target database
opid	*smallint*	Identifies the **dbcc** operation ID
optype	*smallint*	Identifies the **dbcc** operation type
type_code	*int*	Defines the **dbcc** operation type. Valid values are 1000–1007. They are described in Table 12-7.
intvalue	*int*	Specifies the integer value
realvalue	*real*	Specifies the real value
binaryvalue	*varbinary(255)*	Specifies the binary value
stringvalue	*varchar(255)*	Specifies the string value
datevalue	*datetime*	Specifies the date value

Primary key: combination of *dbid*, *opid*, *optype*, and *type_code*

Each "value" column (*intvalue*, *realvalue*, *binaryvalue*, *stringvalue*, and *datevalue*) may contain a null value. At least one is not null. If more than one of these columns contains a value other than null, the columns provide different representations of the same value.

Results of the **dbcc checkstorage** operations include the number of:

- Hard faults found
- Soft faults found
- Operations stopped due to a hard error

dbcc_types

The *dbcc_types* table provides the definitions of the data types used by **dbcc checkstorage**. This table is not actually used by the **dbcc** stored procedures. It is provided to facilitate the use of the other tables in *dbccdb*, and to document the semantics of the data types. Type codes for operation configuration, analysis data reported, fault classification, and fault report parameters are included. If you create your own stored procedures for generating reports, the values listed in the *type_name* column can be used as report headings.

Table 12-7 describes the contents of the *dbcc_types* table. To allow for future additions to *dbcc_types*, some *type_code* numbers are not used at this time.

Table 12-7: Contents of the dbcc_types table

type_code	*type_name*	Description
1	max worker processes	Optional. Specifies the maximum number of worker processes that can be employed. This is also the maximum level of concurrent processing used. Minimum value is 1.
2	dbcc named cache	Specifies the size (in kilobytes) of the cache used by **dbcc checkstorage** and the name of that cache.
3	scan workspace	Specifies the ID and name of the workspace to be used by the database scan.
4	text workspace	Specifies the ID and name of the workspace to be used for text columns.
5	operation sequence number	Specifies the number that identifies the **dbcc** operation that was started most recently.
6	database name	Specifies the name of the database in *sysdatabases*.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
7	OAM count threshold	Specifies the percentage by which the OAM counts must vary before they can be considered to be an error.
8	IO error abort	Specifies the number of I/O errors allowed on a disk before **dbcc** stops checking the pages on that disk.
9	linkage error abort	Specifies the number of linkage errors allowed before **dbcc** stops checking the page chains of an object. Some kinds of page chain corruptions might require a check to be stopped with fewer linkage error than other kinds of page chain corruptions.
1000	hard fault count	Specifies the number of persistent inconsistencies (hard faults) found during the consistency check.
1001	soft fault count	Specifies the number of suspect conditions (soft faults) found during the consistency check.
1002	checks aborted count	Specifies the number of linkage checks that were stopped during the consistency check.
1007	text column count	Specifies the number of non-null *text*/*image* column values found during the consistency check.
5000	bytes data	Specifies (in bytes) the amount of user data stored in the partition being checked.
5001	bytes used	Specifies (in bytes) the amount of storage used to record the data in the partition being checked. The difference between *bytes used* and *bytes data* shows the amount of overhead needed to store or index the data.
5002	pages used	Specifies the number of pages linked to the object being checked that are actually used to hold the object.
5003	pages reserved	Specifies the number of pages that are reserved for the object being checked, but that are not allocated for use by that object. The difference between (8 * *extents used*) and (*pages used* + *pages reserved*) shows the total uncommitted deallocations and pages incorrectly allocated.
5004	pages overhead	Specifies the number of pages used for the overhead functions such as OAM pages or index statistics.
5005	extents used	Specifies the number of extents allocated to the object in the partition being checked. For object 99 (allocation pages), this value is the number of extents that are not allocated to a valid object. Object 99 contains the storage that is not allocated to other objects.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
5006	count	Specifies the number of component items (rows or keys) found on any page in the part of the object being checked.
5007	max count	Specifies the maximum number of component items found on any page in the part of the object being checked.
5008	max size	Specifies the maximum size of any component item found on any page in the part of the object being checked.
5009	max level	Specifies the maximum number of levels in an index. This datatype is not applicable to tables.
5010	pages misallocated	Specifies the number of pages that are allocated to the object, but are not initialized correctly. It is a fault counter.
5011	io errors	Specifies the number of I/O errors encountered. This datatype is a fault counter.
5012	page format errors	Specifies the number of page format errors reported. This datatype is a fault counter.
5013	pages not allocated	Specifies the number of pages linked to the object through its chain, but not allocated. This datatype is a fault counter.
5014	pages not referenced	Specifies the number of pages allocated to the object, but not reached through its chains. This datatype is a fault counter.
5015	overflow pages	Specifies the number of overflow pages encountered. This datatype is only applicable to clustered indexes.
5016	page gaps	Specifies the number of pages not linked to the next page in ascending sequence. This number indicates the amount of table fragmentation.
5017	page extent crosses	Specifies the number of pages that are linked to pages outside of their own extent. As the number of *page extent crosses* increases relative to *pages used* or *extents used*, the effectiveness of large I/O buffers decreases.
5018	page extent gaps	Specifies the number of page extent crosses where the subsequent extent is not the next extent in ascending sequence. Maximal I/O performance on a full scan is achieved when the number of *page extent gaps* is minimized. A seek or full disk rotation is likely for each gap.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
5019	ws buffer crosses	Specifies the number of pages that are linked outside of their workspace buffer cache during the **dbcc checkstorage** operation. This information can be used to size the cache, which provides high performance without wasting resources.
10000	page id	Specifies the location in the database of the page that was being checked when the fault was detected. All localized faults include this parameter.
10001	page header	Specifies the hexadecimal representation of the header of the page that was being checked when the fault was detected. This information is useful for evaluating soft faults and for determining if the page has been updated since it was checked. The server truncates trailing zeros.
10002	text column id	Specifies an 8-byte hexadecimal value that gives the page, row, and column of the reference to a text chain that had a fault. The server truncates trailing zeros.
10003	object id	Specifies a 9-byte hexadecimal value that provides the *object id* (table), the *partition id* (partition of the table) if applicable, and the *index id* (index) of the page or allocation being checked. For example, if a page is expected to belong to table *T1* because it is reached from *T1*'s chain, but is actually allocated to table *T2*, the *object id* for *T1* is recorded, and the *object id expected* for *T2* is recorded. The server truncates trailing zeros.
10007	page id expected	Specifies the page ID that is expected for the linked page when there is a discrepancy between the page ID that is expected and the page ID that is actually encountered. For example, if you follow the chain from *P1* to *P2* when going forward, then, when going backward, *P1* is expected to come after *P2*. The value of *page id expected* is *P1*, and the value of *page id* is *P2*. When the actual value of *P3* is encountered, it is recorded as *page id actual*.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
10008	page id actual	When there is a discrepancy between the page ID that is encountered and the expected page ID, this value specifies the actual page ID that is encountered. (See also, *type_code* 10007.) For example, if you follow the chain from *P1* to *P2* when going forward, then, when going backward, P1 is expected to come after *P2*. The value of *page id expected* is *P1*, and the value of *page id* is *P2*. When the actual value of *P3* is encountered, it is recorded as *page id actual*.
10009	object id expected	Specifies a 9-byte hexadecimal value that provides the expected object id (table), the partition id (partition of the table) if applicable, and the index id (index) of the page or allocation being checked. For example, if a page is expected to belong to table *T1* because it is reached from *T1*'s chain, but is actually allocated to table *T2*, the *object id* for *T1* is recorded, and the *object id expected* for *T2* is recorded. The server truncates trailing zeros.
100000	IO error	Indicates that part of the identified page could not be fetched from the device. This is usually caused by a failure of the operating system or the hardware.
100001	page id error	Indicates that the identifying ID (page number) recorded on the page is not valid. This might be the result of a page being written to or read from the wrong disk location, corruption of a page either before or as it is being written, or allocation of a page without subsequent initialization of that page.
100002	page free offset error	Indicates that the end of data on a page is not valid. This event affects insertions and updates on this page. It might affect some access to the data on this page.
100003	page object id error	Indicates that the page appears to be allocated to some other table than the one expected. If this is a persistent fault, it might be the consequence of either: • An incorrect page allocation, which might only result in the effective loss of this page to subsequent allocation, or • A corrupted page chain, which might prevent access to the data in the corrupted chain
100004	timestamp error	Indicates that the page has a timestamp that is later than the database timestamp. This error can result in failure to recover when changes are made to this page.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
100005	wrong dbid error	Indicates that the database ID *dbid* is stored on the database allocation pages. When this ID is incorrect, the allocation page is corrupt and all the indicated allocations are suspect.
100006	wrong object error	Indicates that the page allocation is inconsistent. The page appears to belong to one table or index, but it is recorded as being allocated to some other table or index in the allocation page. This error differs from *page object id error* in that the allocation is inconsistent, but the consequences are similar.
100007	extent id error	Indicates that an allocation was found for a table or index that is unknown to **dbcc checkstorage**. Typically, this results in the inability to use the allocated storage.
100008	fixed format error	Indicates that the page incorrectly indicates that it contains only rows of a single fixed length. **dbcc checkstorage** reports this error. **dbcc checktable** does not report it, but does repair it.
100009	row format error	Indicates that at least one row on the page is incorrectly formatted. This error might cause loss of access to some or all the data on this page.
100010	row offset error	Indicates that at least one row on the page is not located at the expected page offset. This error might cause loss of access to some or all of the data on this page.
100011	text pointer error	Indicates that the location of the table row that points to the corrupted *text* or *image* data. This information might be useful for correcting the problem.
100012	wrong type error	Indicates that the page has the wrong format. For example, a data page was found in an index or a *text* / *image* column.
100013	non-OAM error	This error is a special case of *wrong type error*. It is not reported as a separate condition in the current release.
100014	reused page error	Indicates that a page is reached by more than one chain and that the chains belong to different objects. This error indicates illegal sharing of a page through corrupt page chain linkages. Access to data in either or both tables might be affected.
100015	page loop error	Indicates that a page is reached a second time while following the page chain for an object, which indicates a loop in the page chain. A loop can result in a session hanging indefinitely while accessing data in that object.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
100016	OAM ring error	Indicates that a page is allocated but not reached by the page chains for the object.Typically, this results in the inability to use the allocated storage.
100017	OAM ring error	Indicates that the OAM page ring linkages are corrupted. This might not affect access to the data for this object, but it might affect insertions, deletions, and updates to that data.
100018	missing OAM error	Indicates that **dbcc checkstorage** found an allocation for the object that was not recorded in the OAM. This error indicates a corruption that might affect future allocations of storage, but probably does not affect access to the presently stored data.
100019	extra OAM error	Indicates that an allocation for this object was recorded in the OAM, but it was not verified in the allocation page. This error indicates a corruption that might affect future allocations of storage, but probably does not affect access to the presently stored data.
100020	check aborted error	Indicates that **dbcc checkstorage** stopped checking the table or index. To prevent multiple fault reports, the check operation on a single chain might be stopped without reporting this error. When an object contains several page chains, failure of the check operation for one chain does not prevent the continuation of the check operation on the other chains unless a fault threshold is exceeded.
100021	chain end error	Indicates that the end of the chain is corrupted. As a soft fault, it might indicate only that the chain was extended or truncated by more than a few pages during the **dbcc checkstorage** operation.
100022	chain start error	Indicates that the start of a chain is corrupted or is not at the expected location. If this is a persistent fault, access to data stored in the object is probably affected.
100023	used count error	Indicates an inconsistency between the count of the pages used that is recorded in the OAM page and the count of the pages used that is determined by examining the allocation pages.
100024	unused count error	Indicates an inconsistency between the count of the pages reserved but unused that is recorded in the OAM page and the count of the pages reserved but unused that is determined by examining the allocation pages.

Table 12-7: Contents of the dbcc_types table (continued)

type_code	*type_name*	Description
100025	row count error	Indicates an inconsistency between the row count recorded in the OAM page and the row count determined by **dbcc checkstorage**.
100026	serialloc error	Indicates a violation of the serial allocation rules applied to log allocations.
100027	text root error	Indicates a violation of the format of the root page of a *text* or *image* index. This check is similar to the root page checks performed by **dbcc textalloc**.
100028	page misplaced	Indicates that pages of this object were not found where they were expected to be from examination of the system tables. This usually indicates that **sp_placeobject** was used sometime in the past. In the *dbcc_counters* table, all misplaced pages are counted together, rather than being reported by device and partition.
100029	page header error	Indicates an internal inconsistency in the page's header other than the kind described by the other type codes. The severity of this error depends on the type of page and the inconsistency found.
100030	page format error	Indicates an internal inconsistency in the page's body other than the kind described by the other type codes. The severity of this error depends on the type of page and the inconsistency found.
100031	page not allocated	Indicates that **dbcc checkstorage** reached an unallocated page by following a page chain. This condition might affect access to data stored in this object.
100032	page linkage error	Indicates that **dbcc checkstorage** detected a fault with either the next or previous linkage of an interior page of a chain. If this is a persistent fault, access to data stored in the object is probably affected.

dbccdb Workspaces

Workspaces are special tables in *dbccdb* that store intermediate results of the **dbcc checkstorage** operation. Workspaces differ from worktables in that they:

- Are preallocated contiguously to improve I/O performance
- Are persistent
- Do not reside in the *tempdb* database

When you create *dbccdb*, two workspaces, *scan* and *text*, are created automatically. They are preallocated as follows:

- **Scan workspace** – contains a row for each page of the target database. The allocation is approximately 1 percent of the database size. Each row consists of a single *binary* (18) column.
- **Text workspace** – contains a row for each table in the target database that contains text or image columns. The size of this table depends on the design of the target database, but it is usually significantly smaller than the *scan* workspace. Each row consists of a single *binary* (22) column.

If either allocation is larger than needed by **dbcc checkstorage**, the operation uses only what it requires. The allocation does not change. If the *text* workspace allocation is too small, **dbcc checkstorage** reports this, recommends a new size, and continues checking; however, not all text chains are checked. If the *scan* workspace allocation is too small, the **dbcc checkstorage** operation fails immediately.

You must have at least one *scan* and one *text* workspace, but you may create as many as you need. While in use, the workspaces are locked so that only one **dbcc checkstorage** operation can use them at any given time. You can execute concurrent **dbcc checkstorage** operations by supplying each one with a separate *scan* and *text* workspace.

For more information on creating workspaces, see the *System Administration Guide* and the *Adaptive Server Reference Manual*.

Ideally, workspaces should be accessed only through **dbcc checkstorage**, but this is not a requirement. **dbcc checkstorage** exclusively locks the workspaces it uses, and the content of the workspaces is regenerated with each execution of **dbcc checkstorage**. The workspaces do not contain any secure data.

➤ ***Note***

While the contents of the workspaces are accessible through SQL, no interpretation of the binary values is available. Access through SQL might return data from different **dbcc** checks mixed together. The presence of a row in these tables does not ensure that it contains valid data. **dbcc** tracks valid rows only during execution. That information is lost when the operation completes.

Most of the update activity in *dbccdb* is performed in the *text* and *scan* workspaces. The workspaces are preallocated, and only one **dbcc checkstorage** operation can use the workspaces at any given time, so

the workspaces are less susceptible to corruption than most user tables. Corruption in a workspace can cause the **dbcc checkstorage** operation to fail or behave erratically. If this happens, drop and re-create the corrupt workspace.

Checks of databases using different workspaces can proceed simultaneously, but the performance of each operation might be degraded as it competes for disk throughput.

To delete a workspace, in *dbccdb*, issue:

```
drop table workspace_name
```

dbccdb Log

The results of each **dbcc checkstorage** operation are recorded in the *dbccdb* log. Updates to the *text* and *scan* workspaces are not recorded there.

The *dbccdb* log must be sized to handle updates to the tables. The log requirement is related to the number of tables and indexes in the target database. It is not related to the target database size.

To minimize the log requirement and the recovery time, use the **truncate log on checkpoint** option with *dbccdb*.

Index

This index provides index entries for all volumes of the *Adaptive Server Reference Manual*. It is divided into three sections:

- Symbols – Indexes entries that begin with symbols.
- Numerics – Indexes entries that begin numerically.
- Subjects – Indexes subjects alphabetically.

Page numbers in **bold** are primary references.

Symbols

Numerics

A

B

C

D

E

F

G

H

J

K

L

M

N

P

Q

R

S

T

U

V

W

X

Y

Z