



Reference Manual Volume 2: Commands

**Adaptive Server Enterprise
12.5**

DOCUMENT ID: 36272-01-1250-01

LAST REVISED: June 2001

Copyright © 1989-2001 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase database management software and to any subsequent release 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.

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. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, the Sybase logo, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, Backup Server, ClearConnect, Client-Library, Client Services, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, E-Anywhere, E-Whatever, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, ImpactNow, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MySupport, Net-Gateway, Net-Library, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, S-Designer, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, Transact-SQL, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server and XP Server are trademarks of Sybase, Inc. 3/01

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

About This Book	vii
CHAPTER 1	Commands 1
	Overview 1
CHAPTER 2	Commands: alter database – continue 7
	alter database 7
	alter role 12
	alter table 16
	begin...end 42
	begin transaction 43
	break 44
	case 45
	checkpoint 48
	close 49
	coalesce 50
	commit 51
	compute Clause 53
	connect to...disconnect 61
	continue 63
CHAPTER 3	Commands: create database – create schema 65
	create database 65
	create default 71
	create existing table 74
	create function (SQLJ) 79
	create index 82
	create plan 97
	create procedure 98
	create procedure (SQLJ) 111
	create proxy_table 114
	create role 116
	create rule 118

	create schema.....	122
CHAPTER 4	Commands: create table – create view.....	125
	create table	125
	create trigger	157
	create view	167
CHAPTER 5	Commands: dbcc – delete statistics.....	177
	dbcc.....	177
	deallocate cursor	186
	declare	186
	declare cursor	188
	delete	193
	delete statistics.....	199
CHAPTER 6	Commands: disk init – disk unmirror	201
	disk init	201
	disk mirror	205
	disk refit.....	208
	disk reinit.....	209
	disk remirror	212
	disk unmirror	214
CHAPTER 7	Commands: drop database – drop view.....	219
	drop database	219
	drop default	220
	drop function (SQLJ).....	221
	drop index	222
	drop procedure.....	223
	drop role	224
	drop rule	225
	drop table	226
	drop trigger.....	228
	drop view.....	229
CHAPTER 8	Commands: dump database – dump transaction	231
	dump database	231
	dump transaction.....	244
CHAPTER 9	Commands: execute – group by having clauses	259
	execute.....	259

	fetch	265
	goto Label	267
	grant	268
	group by and having Clauses.....	280
CHAPTER 10	Commands: if...else – kill	293
	if...else	293
	insert	295
	kill	303
CHAPTER 11	Commands: load database – lock table.....	307
	load database.....	307
	load transaction.....	315
	lock table	324
CHAPTER 12	Commands: nullif – revoke	327
	nullif	327
	online database.....	328
	open	330
	order by Clause.....	331
	prepare transaction	337
	print	337
	quiesce database	340
	raiserror.....	342
	readtext	347
	reconfigure	351
	remove java.....	352
	reorg.....	353
	return	355
	revoke	359
CHAPTER 13	Commands: rollback – save transaction	367
	rollback.....	367
	rollback trigger.....	369
	save transaction	370
CHAPTER 14	Commands: select – setuser	373
	select.....	373
	set	396
	setuser	423

CHAPTER 15	Commands: shutdown – writetext	425
	shutdown	425
	truncate table	427
	union Operator	428
	update	432
	update all statistics	442
	update partition statistics	443
	update statistics	445
	use	448
	waitfor	449
	where Clause	451
	while	458
	writetext	460
Index		463

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 successfully, you need to understand 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, “*System Tables*,” provides reference information about the system tables, which store information about your server, databases, users, and other details of your server. It provides information about the tables in the dbccdb and dbccalt databases.

Audience

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

How to use this book

- Chapter 1, “Commands,” lists the Adaptive Server commands in a table that provides the name and a brief description. Click on a command name in the table to go directly to the command.
- Chapter 2 through Chapter 15 provide manual pages for the individual commands.

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.

Related 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 the Sybase Technical Library.

- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5, 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.
- *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 Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- The *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book. Available only in print version.

- *The System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *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, functions, and stored procedures 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.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using Sybase's DTM XA interface with X/Open XA transaction managers.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Sybase jConnect for JDBC Programmer's Reference* – describes the jConnect for JDBC product and explains how to use it to access data stored in relational database management systems.
- *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Historical Server User's Guide* – describes how to use Historical Server to obtain performance information for SQL Server and Adaptive Server.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.

Other sources of information

- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.

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

- Technical Library CD contains product manuals and is included with your software. The DynaText browser (downloadable from Product Manuals at <http://www.sybase.com/detail/1,3693,1010661,00.html>) 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 the Technical Library.

- Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will 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 Product Manuals Web site, go to Product Manuals at <http://www.sybase.com/support/manuals/>.

Conventions

The following sections describe conventions used in this manual.

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 most 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. Complex commands are formatted using modified Backus Naur Form (BNF) notation.

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 options, utility names, utility options, and other keywords are bold.	<code>select</code> <code>sp_configure</code>
Database names, datatypes, file names and path names are in italics.	<i>master database</i>
Variables, or words that stand for values that you fill in, are in italics.	<code>select <i>column_name</i></code> <code>from <i>table_name</i></code> <code>where <i>search_conditions</i></code>
Type parentheses as part of the command.	<code>compute <i>row_aggregate</i> (<i>column_name</i>)</code>

Element	Example
Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.	<code>::=</code>
Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.	<code>{cash, check, credit}</code>
Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.	<code>[cash check credit]</code>
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.	<code>cash, check, credit</code>
The pipe or vertical bar() means you may select only one of the options shown.	<code>cash check credit</code>
An ellipsis (...) means that you can <i>repeat</i> the last unit as many times as you like.	<pre>buy thing = price [cash check credit] [, thing = price [cash check credit]]...</pre> <p>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.</p>

- 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)

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 the *System Administration Guide*.

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.

This volume describes commands, clauses, and other elements used to construct a Transact-SQL statement.

Overview

Table 1-1 provides a brief description of the commands in this chapter.

Table 1-1: 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 <i>if...else</i> , 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. <i>break</i> is often activated by an <i>if</i> test.
case	Allows SQL expressions to be written for conditional values. <i>case</i> expressions can be used anywhere a value expression can be used.
checkpoint	Writes all <i>dirty</i> 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. <i>coalesce</i> expressions can be used anywhere a value expression can be used; alternative for a <i>case</i> 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.
connect to...disconnect	Specifies the server to which a passthrough connection is required.
continue	Causes the while loop to restart. <i>continue</i> is often activated by an <i>if</i> test.
create database	Creates a new database.

Command	Description
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 <i>column_list</i> , 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.
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.

Command	Description
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.
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.

Command	Description
print	Prints a user-defined message on the user's screen.
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.
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.

Command	Description
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.

Commands: *alter database – continue*

alter database

Description	Increases the amount of space allocated to a database.
Syntax	<pre>alter database <i>database_name</i> [on {default <i>database_device</i> } [= size] [, <i>database_device</i> [= size]]...] [log on { default <i>database_device</i> } [= size] [, <i>database_device</i> [= size]]...] [with override] [for load] [for proxy_update]</pre>
Parameters	<p><i>database_name</i> is the name of the database. The database name can be a literal, a variable, or a stored procedure parameter.</p> <p>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.</p> <p>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:</p> <pre>on default = <i>size</i></pre> <p>To change a database device's status to default, use the system procedure sp_diskdefault.</p> <p><i>database_device</i> 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.</p>

size

is the amount of space to allocate to the database extension. size can be in the following unit specifiers: 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes). Sybase recommend that you always include a unit specifier.

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

Example 1 Adds 1MB to the database mydb on a default database device:

```
alter database mydb
```

Example 2 Adds 3MB to the space allocated for the pubs2 database on the database device named newdata:

```
alter database pubs2
on newdata = 3
```

Example 3 Adds 10MB of space for data on userdata1 and 2MB for the log on logdev:

```
alter database production
on userdata1 = 10
log on logdev = 2
```

Usage

Restrictions

- Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier

- If you do not include a unit specifier, Adaptive Server interprets the size in terms of megabytes of disk space, and this number is converted to the logical page size the server uses.
- Adaptive Server reports an error if the total size of all fixed-length columns, plus the row overhead, is greater than the table's locking scheme and page size allows.
- If you create a DOL table with a variable-length column that exceeds a 8191-byte offset, you cannot add any rows to the column.
- Because Adaptive Server allocates space for databases for create database and alter database in chunks of 256 logical pages, these commands round the specified size down to the nearest multiple of allocation units.
- You can specify the *size* as a float datatype, however, the size is rounded down to the nearest multiple of the allocation unit.
- Although Adaptive Server does create tables in the following circumstances, you will receive errors about size limitations when you perform DML operations:
 - If the total row size for rows with variable-length columns exceeds the maximum column size.
 - If the length of a single variable-length column exceeds the maximum column size.
 - For DOL tables, if the offset of any variable-length column other than the initial column exceeds the limit of 8191 bytes.
- 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.
- Minimum size of a database depends on the logical page size used by the server, described in Table 2-1

Table 2-1: Maximum number of data rows for a DOL data page

Page Size	Maximum number of rows
2K	166
4K	337
8K	678
16K	1361

- 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
```

- 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

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

Description

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
```


Parameters	<p><i>role1</i> is one role in a mutually exclusive relationship.</p> <p>add adds a role in a mutually exclusive relationship; adds a password to a role.</p> <p>drop drops a role in a mutually exclusive relationship; drops a password from a role.</p> <p>exclusive makes both named roles mutually exclusive.</p> <p>membership does not allow you to grant users both roles at the same time.</p> <p>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.</p> <p><i>role2</i> is the other role in a mutually exclusive relationship.</p> <p><i>role_name</i> is the name of the role for which you want to add, drop, or change a password.</p> <p>passwd adds a password to a role.</p> <p><i>password</i> 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.</p> <p>lock locks the specified role.</p> <p>unlock unlocks the specified role.</p> <p>all overrides applies the setting that follows to the entire server rather than to a specific role.</p> <p>set activates the option that follows it.</p>
------------	--

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

Example 1 Defines `intern_role` and `specialist_role` as mutually exclusive:

```
alter role intern_role add exclusive membership
specialist_role
```

Example 2 Defines roles as mutually exclusive at the membership level and at the activation level:

```
alter role specialist_role add exclusive membership
intern_role
alter role intern_role add exclusive activation
surgeon_role
```

Example 3 Adds a password to an existing role:

```
alter role doctor_role add passwd "physician"
```

Example 4 Drops a password from an existing role:

```
alter role doctor_role drop passwd
```

Example 5 Locks the role `physician_role`:

```
alter role physician_role lock
```

Example 6 Unlocks the role `physician_role`:

```
alter role physician_role unlock
```

Example 7 Changes the maximum number of failed logins allowed for `physician_role` to 5:

```
alter role physician_role set max failed_logins 5
```

Example 8 Sets the minimum password length for `physician_role`, an existing role, to five characters:

```
alter role physician_role set min passwd length 5
```

Example 9 Overrides the minimum password length of all roles:

```
alter role "all overrides" set min passwd length -1
```

Example 10 Removes the overrides for the maximum failed logins for all roles:

```
alter role "all overrides" set max failed_logins -1
```

Usage

- 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

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

Description

Adds new columns to a table; drops or modifies existing columns; adds, changes, or drops constraints; changes properties of an existing table; enables or disables triggers on a table.

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]
  [, next_column]...

| 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

```

Parameters

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 *tablename_colname_objid*, where *tablename* 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 16K bytes, depending on the page size of the database server and other variables. 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 *tablename_colname_objectid*, where *tablename* 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 *tablename_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

Example 1 Adds a column to a table. For each existing row in the table, Adaptive Server assigns a NULL column value:

```
alter table publishers
add manager_name varchar(40) null
```

Example 2 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:

```
alter table sales_daily
add ord_num numeric(5,0) identity
```

Example 3 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):

```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
```

Example 4 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:

```
alter table authors
add constraint au_identification
```

```
primary key (au_id, au_lname, au_fname)
with reservepagegap = 16
```

Example 5 Drops the `au_identification` constraint:

```
alter table titles
drop constraint au_identification
```

Example 6 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:

```
alter table authors
replace phone default null
```

Example 7 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:

```
alter table titleauthor partition 5
```

Example 8 Concatenates all page chains of the titleauthor table, then repartitions it with six partitions:

```
alter table titleauthor unpartition
alter table titleauthor partition 6
```

Example 9 Changes the locking scheme for the titles table to datarows locking:

```
alter table titles lock datarows
```

Example 10 Adds the not-null column `author_type` to the authors table with a default of `primary_author`:

```
alter table authors
add author_type varchar(20)
default "primary_author" not null
```

Example 11 Drops the `advance`, `notes`, and `contract` columns from the titles table:

```
alter table titles
drop advance, notes, contract
```

Example 12 Modifies the `city` column of the authors table to be a `varchar(30)` with a default of NULL:

```
alter table authors
modify city varchar(30) null
```

Example 13 Modifies the `stor_name` column of the `stores` table to be NOT NULL. Note that its datatype, `varchar(40)`, remains unchanged:

```
alter table stores
  modify stor_name not null
```

Example 14 Modifies the `type` column of the `titles` table and changes the locking scheme of the `titles` table from `allpages` to `datarows`:

```
alter table titles
  modify type varchar(10)
  lock datarows
```

Example 15 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:

```
alter table titles
  modify notes varchar(150) not null
  with exp_row_size = 40
```

Example 16 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:

```
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
```

Usage

- 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.
- When the table owner uses `alter table`, Adaptive Server disables access rules during the execution of the command and enables them upon completion of the command. The access rules are disabled to avoid filtering of the table data during `alter table`.

Restrictions

- You cannot add a column of datatype `bit` to an existing table.
- The maximum number of columns in a table is:
 - 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.

- 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.
- 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.
- alter table raises an error if the number of variable-length columns in an APL table exceeds 254.

Warning! Do not alter the system tables.

- The maximum length for in row Java columns is determined by the maximum size of a variable-length column for the table’s schema, locking style, and page size.
- When converting a table to a different locking scheme, the data in the source table cannot violate the limits of the target table. For example, if you attempt to convert a DOL with more than 254 variable-length columns to an APL table, alter table fails because an APL table is restricted to having no more than 254 columns.
- Columns with fixed-length data (for example char, binary, and so on) have the maximum sizes shown in Table 2-2:

Table 2-2: Maximum row and column length - APL & DOL

Locking scheme	Page size	Maximum row length	Maximum column length
	2K (2048 bytes)	1962	1960 bytes
	4K (4096 bytes)	4010	4008 bytes
APL tables	8K (8192 bytes)	8106	8104 bytes
	16K (16384 bytes)	16298	16296 bytes
	2K (2048 bytes)	1964	1958 bytes
	4K (4096 bytes)	4012	4006 bytes
DOL tables	8K (8192 bytes)	8108	8102 bytes
	16K (16384 bytes)	16300	16294 bytes if table does not include any variable length columns
	16K (16384 bytes)	16300 (subject to a max start offset of varlen = 8191)	8191-6-2 = 8183 bytes if table includes at least on variable length column.*
* This size includes six bytes for the row overhead and two bytes for the row length field			

- The maximum number of bytes of variable length data per row depends on the locking scheme for the table. Table 2-3 describes the maximum size columns for an APL table and Table 2-4 describes the maximum size columns for a DOL table:

Table 2-3: Maximum size for variable-length columns in an APL table

Page size	Maximum row length	Maximum column length
2K (2048 bytes)	1962	1948
4K (4096 bytes)	4010	3988
8K (8192 bytes)	8096	8058
16K (16384 bytes)	16298	16228

Table 2-4: Maximum size for variable-length columns in an DOL table

Page size	Maximum row length	Maximum column length
2K (2048 bytes)	1964	1954
4K (4096 bytes)	4012	4002
8K (8192 bytes)	8108	7998
16K (16384 bytes)	16300	162290

- 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.
- The maximum value for max_rows_per_page is 256 bytes for APL tables. max_rows_per_page parameter is not used for DOL 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.
- You cannot use alter table to add a declarative or check constraint and then insert data into the table in the same batch or procedure. Either separate the alter and insert statements into two different batches or procedures, or use execute to perform the actions separately.
- You cannot use the following variable in alter table statements that include defaults:

```
declare @a int
select @a = 2
```

```
alter table t2 add c3 int
default @a
```

Doing so results in error message 154, which says, “Variable is not allowed in default.”

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 Chapter 8, “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 2-5: Information stored about referential integrity constraints

Information stored in <code>sysreferences</code>	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, to change the `exp_row_size` value for a table, 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 39 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 2-6 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 2-6: 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 2-7 shows the default values and the effects of using the default values for the space management properties.

Table 2-7: 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
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 2-8 shows how the values are converted.

Table 2-8: 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 <i>exp_row_size</i> percent
255	1, that is, fully packed pages
1–254	The smaller of: <ul style="list-style-type: none"> • maximum row size • $2002/\text{max_rows_per_page}$ 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 2-9:

Table 2-9: When can I add, drop, or modify a column in data-only locked table?

Type of index	All pages Locked, partitioned table	Allpages Locked, unpartitioned table	Data-only locked, partitioned table	Data-only locked, partitioned table
Clustered	Yes	Yes	No	Yes
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:

- Drop the clustered index.
 - Alter the (data-only locked) table.
 - 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 can drop columns that have defaults or rules bound to them. Any column-specific defaults are also dropped when you drop the column. You cannot drop columns that have check constraints or referential constraints bound to them. Instead, first drop the check constraint or referential constraint, 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 table's 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
 - a Drop the clustered index.
 - b Alter the table's schema.
 - c 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
 - unichar
 - univarchar
 - 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 16K bytes, depending on the page size of the data server.
- 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

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_helppartition, sp_rename

begin...end

Description 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

Parameters *statement block*
is a series of statements enclosed by begin and end.

Examples **Example 1** Without begin and end, the if condition would cause execution of only one SQL statement:

```
if (select avg(price) from titles) < $15
begin
    update titles
    set price = price * $2
    select title, price
    from titles
    where price > $28
end
```

Example 2 Without begin and end, the print statement would not execute:

```
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."
```

Usage

- begin...end blocks can nest within other begin...end blocks.

Standards

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

Description

Marks the starting point of a user-defined transaction.

Syntax

```
begin tran[saction] [transaction_name]
```

Parameters

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

Explicitly begins a transaction for the insert statement:

```
begin transaction
  insert into publishers (pub_id) values ("9999")
commit transaction
```

Usage

- 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

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

Description

Causes an exit from a while loop. break is often activated by an if test.

Syntax

```
while logical_expression
    statement
break
    statement
continue
```

Parameters

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

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:

```
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
```

Usage

- 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

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

Description

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
```

Parameters

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” on page 179.

Examples

Example 1 Selects all the authors from the authors table and, for certain authors, specifies the city in which they live:

```
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
```

Example 2 Returns the first occurrence of a non-NULL value in either the lowqty or highqty column of the discounts table:

```
select stor_id, discount,
       coalesce (lowqty, highqty)
from discounts
```

Example 3 This is an alternative way of writing example 2:

```
select stor_id, discount,
       case
         when lowqty is not NULL then lowqty
         else highqty
       end
from discounts
```

Example 4 Selects the *titles* and *type* from the *titles* table. If the book type is UNDECIDED, nullif returns a NULL value:

```
select title,
       nullif(type, "UNDECIDED")
from titles
```

Example 5 This is an alternative way of writing example 4:

```
select title,
       case
         when type = "UNDECIDED" then NULL
```

```

        else type
      end
    from titles

```

Usage

- 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 expression must be something other than the null keyword. For example:

```

select price,
       coalesce (NULL, NULL, NULL)
from titles

```

produces 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

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

Description Writes all **dirty** pages (pages that have been updated since they were last written) to the database device.

Syntax checkpoint

Examples Writes all dirty pages in the current database to the database device, regardless of the system checkpoint schedule:

```
checkpoint
```

Usage

- 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

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

Description

Deactivates a cursor.

Syntax

close *cursor_name*

Parameters

cursor_name
is the name of the cursor to close.

Examples

Closes the cursor named authors_crsr:

```
close authors_crsr
```

Usage

- 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

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

Description Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a case expression.

Syntax `coalesce(expression, expression [, expression]...)`

Parameters `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” on page 179.

Examples **Example 1** Returns the first occurrence of a non-NULL value in either the lowqty or highqty column of the discounts table:

```
select stor_id, discount,
       coalesce (lowqty, highqty)
from discounts
```

Example 2 This is an alternative way of writing example 1:

```
select stor_id, discount,
       case
           when lowqty is not NULL then lowqty
           else highqty
       end
from discounts
```

Usage

- `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

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

Description

Marks the ending point of a user-defined transaction.

Syntax

`commit [tran | transaction | work] [transaction_name]`

Parameters

`tran | transaction | work`

specifies that you want to commit the transaction or the work. If you specify `tran`, `transaction`, or `work`, you can also specify the *transaction_name*.

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

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:

```
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
```

Usage

- 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

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

Description

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]]...
```

Parameters

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

Example 1 Calculates the sum of the prices of each type of cook book that costs more than \$12:

```
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
trad_cook	14.99
trad_cook	20.95
	sum
	35.94

(5 rows affected)

Example 2 Calculates the sum of the prices and advances for each type of cook book that costs more than \$12:

```
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)

Example 3 Calculates the sum of the prices and maximum advances of each type of cook book that costs more than \$12:

```
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)

Example 4 Breaks on type and pub_id and calculates the sum of the prices of psychology books by a combination of type and publisher ID:

```
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)

```

Example 5 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:

```

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)

```

Example 6 Calculates the grand totals of the prices and advances of cook books that cost more than \$10:

```
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)

Example 7 Calculates the sum of the price of cook books and the sum of the price used in an expression:

```
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

Usage

- 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.
- If a `compute` clause includes a `group by` clause:
 - The `compute` clause cannot contain more than 255 aggregates
 - The `group by` clause cannot contain more than 255 columns
- Columns included in a `compute` clause cannot be longer than 255 bytes

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:

```
-----
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 2-10 lists the output and grouping of different types of compute clauses.

Table 2-10: 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

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

Description	(Component Integration Services only) Connects to the specified server and disconnects the connected server.
Syntax	connect to <i>server_name</i> disconnect
Parameters	<i>server_name</i> is the server to which a passthrough connection is required.
Examples	<p>Example 1 Establishes a passthrough connection to the server named SYBASE:</p> <pre>connect to SYBASE</pre> <p>Example 2 Disconnects the connected server:</p> <pre>disconnect</pre>

Usage

- 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 sys.servers 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 sys.attributes, 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

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

Description

Restarts the while loop. continue is often activated by an if test.

Syntax

```
while boolean_expression
    statement
    break
    statement
continue
```

Examples

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:

```
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
```

continue

Usage

- *continue* restarts the while loop, skipping any statements after *continue*.

Standards

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*

Commands: *create database – create schema*

create database

Description	Creates a new database.
Syntax	<pre>create database <i>database_name</i> [on {default <i>database_device</i>} [= <i>size</i>] [, <i>database_device</i> [= <i>size</i>]]...] [log on <i>database_device</i> [= <i>size</i>] [, <i>database_device</i> [= <i>size</i>]]...] [with {override default_location = "pathname"}] [for {load proxy_update}]</pre>
Parameters	<p><i>database_name</i> is the name of the new database. It must conform to the rules for identifiers and cannot be a variable.</p> <p>on indicates a location and size for the database.</p> <p>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:</p> <pre>on default = <i>size</i></pre> <p>To change a database device's status to "default," use the system procedure sp_diskdefault.</p> <p><i>database_device</i> 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.</p>

size

is the amount of space to allocate to the database extension size can be in the following unit specifiers: 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.

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. If 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" on page 69 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

Example 1 Creates a database named pubs:

```
create database pubs
```

Example 2 Creates a 4MB database named pubs:

```
create database pubs  
on default = 4
```

Example 3 Creates a database named pubs with 3MB on the datadev segment and 2 MB on the moredatadev segment:

```
create database pubs
```

```
on datadev = 3, moredatadev = 2
```

Example 4 Creates a database named pubs with 3MB of data on the datadev segment and a 1MB log on the logdev segment:

```
create database pubs
on datadev = 3
log on logdev = 1
```

Example 5 Creates a proxy database named proxydb but does not automatically create proxy tables:

```
create database proxydb
with default_location
"UNITEST.pubs.dbo."
```

Example 6 Creates a proxy database named proxydb and automatically creates proxy tables:

```
create database proxydb
on default = 4
with default_location
"UNITEST.pubs2.dbo."
for proxy_update
```

Usage

- Use create database from the master database.
- You can specify the *size* as a float datatype, however, the size is rounded down the nearest multiple of the allocation unit.
- Because Adaptive Server allocates space for databases for create database and alter database in chunks of 256 logical pages, these commands round the specified size down to the nearest multiple of allocation units
- If you do not include a unit specifier, Adaptive Server interprets the size in terms of megabytes of disk space, and this number is converted to the logical page size the server uses.
- 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
```

- 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

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_helppdb, sp_logdevice, sp_renamedb, sp_spaceused

create default

Description

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
```

Parameters

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

Example 1 Defines a default value. Now, you need to bind it to the appropriate column or user-defined datatype with `sp_bindefault`:

```
create default phonedflt as "UNKNOWN"  
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.

Example 2 Creates a default value, `today's_date`, that inserts the current date into the columns to which it is bound:

```
create default today's_date as getdate()
```

Usage

- 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 text of a default, use `sp_hidextext`.

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_truncation* 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 3-1 illustrates the relationship between the existence of a default and the definition of a column as NULL or NOT NULL.

Table 3-1: 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

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

Description	(Component Integration Services only) 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]
Parameters	<p><i>table_name</i> specifies the name of the table for which you want to create a proxy table.</p> <p><i>column_list</i> specifies the name of the column list that stores information about the remote table.</p> <p>on <i>segment_name</i> specifies the segment that contains the remote table.</p> <p>external specifies that the object is a remote object.</p> <p>table specifies that the remote object is a table or a view. The default is external table.</p> <p>procedure specifies that the remote object is a stored procedure.</p>

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

Example 1 Creates the proxy table authors:

```
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
)
```

Example 2 Creates the proxy table syb_columns:

```
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
)
```

)

Example 3 Creates a proxy table named `blurbs` for the `blurbs` table at the remote server `SERVER_A`:

```
create existing table blurbs
(author_id          id          not null,
 copy              text        not null)
at "SERVER_A.db1.joe.blurbs"
```

Example 4 Creates a proxy table named `rpc1` for the remote procedure named `p1`:

```
create existing table rpc1
(column_1          int,
 column_2          int)
external procedure
at "SERVER_A.db1.joe.p1"
```

Usage

- 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.sybsemprocs.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
```

Component Integration Services 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

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 function (SQLJ)

Description

Creates a user-defined function by adding a SQL wrapper to a Java static method. Can return a value defined by the method.

Syntax

```
create function [owner.]sql_function_name
    ( [ sql_parameter_name sql_datatype
      [( length)| (precision[, scale ] ) ]
      [ [, sql_parameter_name sql_datatype
```

```

        [( length ) | ( precision [, scale ] ) ]
    ... ] ] )
returns sql_datatype
    [ ( length ) | ( precision [, scale ] ) ]
[modifies sql data]
[returns null on null input |
 called on null input]
[deterministic | not deterministic]
[exportable]
language java
parameter style java
external name 'java_method_name
    [ ( [java_datatype [, java_datatype
    ... ] ] ) ]'

```

Parameters

sql_function_name

is the Transact-SQL name of the function. It must conform to the rules for identifiers and cannot be a variable.

sql_parameter_name

is the name of an argument to the function. The value of each input parameter is supplied when the function is executed. Parameters are optional; a SQLJ function need not take arguments.

Parameter names must conform to the rules for identifiers. 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 the parameter begins with a numeric character, it also must be enclosed in quotes.

sql_datatype [(length) | (precision [, scale])]

is the Transact-SQL datatype of the parameter. See create procedure on page 98 for more information about these parameters.

sql_datatype is the SQL procedure signature.

returns *sql_datatype*

specifies the result datatype of the function.

modifies sql data

indicates that the Java method invokes SQL operations, reads, and modifies SQL data in the database. This is the default and only implementation. It is included for syntactic compatibility with the ANSI standard.

deterministic | not deterministic

supported for syntactic compatibility with the ANSI standard. Not implemented.

exportable

specifies that the procedure is to be run on a remote server using the Adaptive Server OmniConnect™ feature. Both the procedure and the method it is built on must reside on the remote server.

language java

specifies that the external routine is written in Java. This is a required clause for SQLJ functions.

parameter style java

specifies that the parameters passed to the external routine at runtime are Java parameters. This is a required clause for SQLJ functions.

external

indicates that create function defines a SQL name for an external routine written in a programming language other than SQL.

name

specifies the name of the external routine (Java method). The specified name—'*java_method_name* [*java_datatype*{[, *java_datatype*} ...]}'—is a character-string literal and must be enclosed in single quotes.

java_method_name

specifies the name of the external Java method.

java_datatype

specifies a Java datatype that is mappable or result-set mappable. This is the Java method signature.

Examples

This example creates a function `square_root` that invokes the `java.lang.Math.sqrt()` method:

```
create function square_root
  (input_number double precision) returns
  double precision
  language java parameter style java
  external name 'java.lang.Math.sqrt'
```

Usage

- You cannot create a SQLJ function with the same name as an Adaptive Server built-in function.

- You can create user-defined functions (based on Java static methods) and SQLJ functions with the same class and method names.

Note Adaptive Server's searching order ensures that the SQLJ function is always found first.

- You can include a maximum of 31 parameters in a create function statement.
- See *Java in Adaptive Server Enterprise* for more information about create function.

Permissions

Only the Database Owner or user with sa role can execute create function. The Database Owner or sa cannot transfer permission for create function.

See also

Commands – create function (SQLJ), drop function (SQLJ)

System procedures – sp_depends, sp_help, sp_helpjava, sp_helprotect

create index

Description

Creates an index on one or more columns in a table.

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]
```

Parameters

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” on page 90.

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 value you specify is not saved in sysindexes for display by sp_helpindex or for later use by the reorg command. Use sp_chgattribute to create stored fillfactor values.

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

cancel 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” on page 92 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” on page 92 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” on page 93 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

Example 1 Creates an index named au_id_ind on the au_id column of the authors table:

```
create index au_id_ind
on authors (au_id)
```

Example 2 Creates a unique clustered index named au_id_ind on the au_id column of the authors table:

```
create unique clustered index au_id_ind
on authors(au_id)
```

Example 3 Creates an index named ind1 on the au_id and title_id columns of the titleauthor table:

```
create index ind1
on titleauthor (au_id, title_id)
```

Example 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:

```
create nonclustered index zip_ind
on authors(postalcode)
with fillfactor = 25, consumers = 4
```

Example 5 Creates an index with ascending ordering on pub_id and descending order on pubdate:

```
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
```

Example 6 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:

```
create index title_id_ix
    on titles (title_id)
    with reservepagegap = 40,
    statistics using 50 values
```

Usage

- 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.
- You can create non-indexable columns in a table by creating columns with lengths greater than the index row-size limit. See for more information:

Logical page size	Index row-size limit
2K	650
4K	1310
8K	2670
16K	5390

- 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 24, “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 24, “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 8, "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 3-4. 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 3-2 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 3-2: 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.
<code>allow_dup_row</code> set	create index completes.	insert completes.
<code>ignore_dup_row</code> set	Index is created but duplicate rows are deleted; error message.	All rows are inserted except duplicates; error message.

Table 3-3 shows which index options can be used with the different types of indexes:

Table 3-3: 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 3-4 shows when the sort is required and when the table is copied for partitioned and nonpartitioned tables.

Table 3-4: 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 3-5 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 3-5: 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 3-6 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 3-6: 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

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions	create index permission defaults to the table owner and is not transferable.
See also	<i>Commands</i> – alter table, create table, drop index, insert, order by Clause, set, update <i>System procedures</i> – sp_addsegment, sp_chgattribute, sp_helpindex, sp_helpsegment, sp_spaceused <i>Utilities</i> – optdiag

create plan

Description	Creates an abstract plan.
Syntax	<pre>create plan <i>query plan</i> [into <i>group_name</i>] [and set @<i>new_id</i>]</pre>
Parameters	<p><i>query</i> is a string literal, parameter, or local variable containing the SQL text of a query.</p> <p><i>plan</i> is a string literal, parameter, or local variable containing an abstract plan expression.</p> <p>into <i>group_name</i> specifies the name of an abstract plan group.</p> <p>and set @ <i>new_id</i> returns the ID number of the abstract plan in the variable.</p>
Examples	<p>Example 1 Creates an abstract plan for the specified query:</p> <pre>create plan "select * from titles where price > \$20" "(t_scan titles)"</pre> <p>Example 2 Creates an abstract plan for the query in the dev_plans group, and returns the plan ID in the variable @<i>id</i>:</p> <pre>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</pre>

Usage	<ul style="list-style-type: none">• 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	create plan is a Transact-SQL extension.
Permissions	create plan permission defaults to all users. No permission is required to use it.
See also	<i>Commands</i> – set plan <i>System procedures</i> – sp_add_qpgroup, sp_find_qplan, sp_help_qpgroup, sp_set_qplan

create procedure

Description	Creates a stored procedure or an extended stored procedure (ESP) that can take one or more user-supplied parameters.
-------------	--

Note For syntax and usage information about the SQLJ command for creating procedures, see create function (SQLJ) on page 79.

Syntax	<pre>create procedure [owner.]procedure_name[:number] [(<i>@parameter_name</i> datatype [(length) (precision [, scale])] [= default][output] [, <i>@parameter_name</i> datatype [(length) (precision [, scale])] [= default][output]...{<i>}]</i>)]</pre>
--------	---

[with recompile]
as {*SQL_statements* | external name *dll_name*}

Parameters***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 following statement is not allowed:

```
drop procedure orderproc;2
```

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, unichar, univarchar, 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

Example 1 Given a table name, the procedure `showind` displays its name and the names and identification numbers of any indexes on any of its columns:

```
create procedure showind @tablename varchar(30)
as
    select sysobjects.name, sysindexes.name, indid
    from sysindexes, sysobjects
    where sysobjects.name = @tablename
    and sysobjects.id = sysindexes.id
```

Here are the acceptable syntax forms for executing `showind`:

```
execute showind titles
execute showind @tablename = "titles"
```

Or, if this is the first statement in a file or batch:

```
showind titles
```

Example 2 This procedure displays information about the system tables if the user does not supply a parameter:

```
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
```

Example 3 This procedure specifies an action to be taken if the parameter is NULL (that is, if the user does not give a parameter):

```
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
```

Example 4 This procedure multiplies two integer parameters and returns the product in the output parameter, *@result*:

```
create procedure mathtutor @mult1 int, @mult2 int,
    @result int output
```

```
as
select @result = @mult1 * @mult2
```

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)
```

Example 5 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:

```
declare @guess int
select @guess = 32
exec mathtutor 5, 6, @result = @guess output

(1 row affected)
(return status = 0)
```

Return parameters:

```
@result
-----
      30
```

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*).

Example 6 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.

```
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!
```

Example 7 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`:

```
create procedure xp_echo @in varchar(255),
    @out varchar(255) output
as external name "sqlsrvdll.dll"
```

Usage

- 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
-----
sybssystemprocs
    
```

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.

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 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.

Standards

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 at procedure creation 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 at procedure execution 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 procedure (SQLJ)

Description Creates a SQLJ stored procedure by adding a SQL wrapper to a Java static method. Can accept user-supplied parameters and return result sets and output parameters.

Note For syntax and usage information about the Transact-SQL command for creating procedures, see “create procedure” on page 98.

Syntax

```
create procedure [owner.]sql_procedure_name
  ([ [ in | out | inout ] sql_parameter_name
    sql_datatype [( length) |
    (precision[, scale]) ]
  [, [ in | out | inout ] sql_parameter_name
    sql_datatype [( length) |
    (precision[, scale]) ] ]
  ...)
  [modifies sql data ]
  [dynamic result sets integer]
  [deterministic | not deterministic]
  language java
  parameter style java
  external name 'java_method_name
  [ ( [java_datatype[, java_datatype
  ...]) ] ]'
```

Parameters

sql_procedure_name
is the Transact-SQL 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.

in | out | inout
specifies the mode of the listed parameter. *in* indicates an input parameter; *out* indicates an output parameter; and *inout* indicates a parameter that is both an input and an output parameter. The default mode is *in*.

sql_parameter_name

is the name of an argument to the procedure. The value of each input parameter is supplied when the procedure is executed. Parameters are optional; a SQLJ stored procedure need not take arguments.

Parameter names must conform to the rules for identifiers. 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 the parameter begins with a numeric character, it also must be enclosed in quotes.

sql_datatype [(length) | (precision [, scale])]

Is the Transact-SQL datatype of the parameter.

sql_datatype is the SQL procedure signature.

modifies sql data

Indicates that the Java method invokes SQL operations, reads, and modifies SQL data in the database. This is the default and only implementation. It is included for syntactic compatibility with the ANSI standard.

dynamic result sets *integer*

Specifies that the Java method can return SQL result sets. *integer* specifies the maximum number of result sets the method can return. This value is implementation-defined.

deterministic | not deterministic

This syntax is supported for compatibility with other SQLJ-compliant vendors.

language java

Specifies that the external routine is written in Java. This is a required clause for SQLJ stored procedures.

parameter style java

Specifies that the parameters passed to the external routine at runtime are Java parameters. This is a required clause for SQLJ stored procedures.

external

Indicates that create procedure defines a SQL name for an external routine written in a programming language other than SQL.

name

Specifies the name of the external routine (Java method). The specified name—'*java_method_name* [*java_datatype*{[, *java_datatype*} ...]}'—is a character-string literal and must be enclosed in single quotes.

java_method_name

Specifies the name of the external Java method.

java_datatype

Specifies a Java datatype that is mappable or result-set mappable. This is the Java method signature.

Examples

This example creates the SQLJ procedure `java_multiply`, which multiplies two integers and returns an integer.

```
create procedure java_multiply (param1 integer,
                               param2 integer, out result integer)
  language java parameter style java
  external name 'MathProc.multiply'
```

Usage

- You can include a maximum of 31 in, inout, and out parameters in a create procedure statement.
- To comply with the ANSI standard, you do not precede parameter names with the @ sign. When executing a SQLJ stored procedure from isql or other non-Java client, however, you must precede parameter names with the @ sign, which preserves the naming order.
- The SQLJ create procedure syntax differs from the Transact-SQL create procedure syntax for compatibility with the SQLJ ANSI standard. Adaptive Server executes each type of stored procedure in the same way.

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.

See also

Commands – create function (SQLJ), drop procedure

System procedures – sp_depends, sp_help, sp_helpjava, sp_helprotect

create proxy_table

Description	<i>Component Integration Services only</i> – 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	<pre>create proxy_table table_name [external [table directory file]] at pathname</pre>
Parameters	<p><i>table_name</i> specifies the local proxy table name to be used by subsequent statements. <i>table_name</i> takes the form:</p> <p>dbname.owner.object</p> <p>where <i>dbname</i> and <i>owner</i> are optional and represent the local database and owner name. If <i>dbname</i> is not specified, the table is created in the current database; if <i>owner</i> is not specified, the table is owned by the current user. If either <i>dbname</i> or <i>owner</i> is specified, the entire <i>table_name</i> must be enclosed in quotes. If only <i>dbname</i> is present, a placeholder is required for <i>owner</i>.</p> <p>external table specifies that the object is a remote table or view. external table is the default, so this clause is optional.</p> <p>external directory specifies that the object is a directory with a path in the following format: <code>"/tmp/directory_name [;R]"</code>. The option “R” indicates recursive.</p> <p>external file specifies that the object is a file with a path in the following format: <code>"/tmp/filename"</code>.</p>

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

This example creates a proxy table named t1 that is mapped to the remote table t1. Component Integration Services derives the column list from the remote table:

```
create proxy_table t1
at "SERVER_A.db1.joe.t1"
```

Usage

- 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. Component Integration Services 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.
- Component Integration Services 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

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

Description

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 ] ]
```

Parameters

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.

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

Example 1 Creates a role named doctor_role:

```
create role doctor_role
```

Example 2 Creates a role named doctor_role with the password physician:

```
create role doctor_role with passwd "physician"
```

Example 3 Sets the password expiration for intern_role:

```
create role intern_role, with passwd "temp244",  
passwd expiration 7
```

Example 4 Sets the maximum number of failed logins allowed for intern_role:

```
create role intern_role with passwd "temp244",  
max failed_logins 20
```

Example 5 Sets the minimum password length for intern_role:

```
create role intern_role with passwd "temp244",  
min passwd length 0
```

Usage

- 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

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

Description

Specifies the domain of acceptable values for a particular column or for any column of a user-defined datatype and creates access rules.

Syntax

```
create rule [ [ and | or ] access]
           [owner.]rule_name
           as condition_expression
```

Parameters

access

specifies that you are creating an access rule. For information on access rules, see Chapter 11, “Managing User Permissions” in the *System Administration Guide*.

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

Example 1 Creates a rule named *limit* which limits the value of *advance* to less than \$1000:

```
create rule limit
as @advance < $1000
```

Example 2 Creates a rule named *pubid_rule* which restricts the values of *pub_id* to 1389, 0736, or 0877:

```
create rule pubid_rule
as @pub_id in ('1389', '0736', '0877')
```

Example 3 Creates a rule named *picture* which restricts the value of *value* to always begin with the indicated characters:

```
create rule picture
as @value like '[_-%][0-9]'
```

Usage

- 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 3-7: 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.

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.

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.

Standards

Standard	Compliance Level	Comments
SQL92	Transact-SQL extension	To create rules using SQL92-compliant syntax, use the <i>check</i> clause of the <i>create table</i> 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 rule, drop table
System procedures – sp_bindrule, sp_help, sp_helptext, sp_hidetext, sp_rename, sp_unbindrule

create schema

Description Creates a new collection of tables, views, and permissions for a database user.

Syntax
`create schema authorization authorization_name
 create_object_statement
 [create_object_statement ...]
 [permission_statement ...]`

Parameters
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 Creates the newtitles, newauthors, newtitleauthors tables, the tit_auth_view view, and the corresponding permissions:

```
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
    
```

Usage

- 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

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*
Utilities – *isql*

Commands: *create table* – *create view*

create table

Description

Creates new tables and optional integrity constraints.

Syntax

```
create table [database .]owner .]table_name (column_name datatype
[default {constant_expression | user | null}]
{{{identity | null | not null}}
[off row | [ in row [ (size_in_bytes) ] ]
[[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) ... }
[column_name | next_constraint}]...])
[lock {datarows | datapages | allpages } ]
[with { max_rows_per_page = num_rows,
exp_row_size = num_bytes,
reservepagegap = num_pages,
identity_gap = value } ]
[on segment_name ]
[ [ external table ] at pathname ]
```

Parameters

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 7, "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 7, "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" on page 190.

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 *tablename_colname_objid*, where *tablename* 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 default value is off row. For more information, see *Java in Adaptive Server Enterprise*.

size_in_bytes

specifies the maximum size of the in-row column. An object stored in-row can occupy up to approximately 16K bytes, depending on the page size of the database server and other variables.

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 *tablename_colname_objectid* where *tablename* 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 *tablename_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.

with identity_gap

specifies the identity gap for the table. This value overrides the system identity gap setting for this table only.

value

is the identity gap amount.

For more information about setting the identity gap, see *IDENTITY* columns.

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

Example 1 Creates the titles table:

```
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)
```

Example 2 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:

```
create table "compute"  
("max" int, "min" int, "total score" int)
```

Example 3 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.):

```
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))
```

Example 4 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:

```
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))
```

Example 5 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_idrulerule` 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]"
```

```
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)
```

Example 6 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:

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

Example 7 Specifies the `datapages` locking scheme for the `new_titles` table and an expected row size of 200:

```
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
```

Example 8 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:

```
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
```

Example 9 Creates a constraint supported by a unique clustered index; the index order is ascending for stor_id and descending for ord_num:

```
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))
```

Example 10 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:

```
create table t1
(a          int,
b          char(10))
at "SERVER_A.db1.joe.t1"
```

Example 11 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:

```
create table employees
(name varchar(30),
home_addr Address,
mailing_addr Address2Line)
```

Example 12 Creates a table named mytable with an identity column. The identity gap is set to 10, which means ID numbers are allocated in memory in blocks of ten. If the server fails or is shut down with no wait, the maximum gap between the last ID number assigned to a row and the next ID number assigned to a row is ten numbers:

```
create table mytable
(IdNum numeric(12,0) identity)
with identity_gap = 10
```

Example 13 Creates a table named mytable with an identity column. The identity gap is set to 10, which means ID numbers will be allocated in memory in blocks of ten. If the server fails or is shut down with no wait, the maximum gap between the last ID number assigned to a row and the next ID number assigned to a row is ten numbers:

```
create table mytable
(IdNum numeric(12,0) identity)
with identity_gap = 10
```

For more information about identity gaps, see the section “Managing Identity Gaps in Tables” in Chapter 7, “Creating Databases and Tables” in the *Transact-SQL User’s Guide*.

Usage

- 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 2,000,000,000 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 columns you can create in a table is:
 - 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.
 - 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.
 - 1024 for fixed-length columns in both all-pages-locked (APL) and data-only-locked (DOL) tables.
- Although Adaptive Server does create tables in the following circumstances, you will receive errors about size limitations when you perform DML operations:
 - If the total row size for rows with variable-length columns exceeds the maximum column size.
 - If the length of a single variable-length column exceeds the maximum column size.
 - For DOL tables, if the offset of any variable-length column other than the initial column exceeds the limit of 8191 bytes.

- Adaptive Server reports an error if the total size of all fixed-length columns, plus the row overhead, is greater than the table's locking scheme and page size allows. These limits are described in Table 4-1.

Table 4-1: Maximum row and column length - APL & DOL

Locking scheme	Page size	Maximum row length	Maximum column length
APL tables	2K (2048 bytes)	1962	1960 bytes
	4K (4096 bytes)	4010	4008 bytes
	8K (8192 bytes)	8106	8104 bytes
	16K (16384 bytes)	16298	16296 bytes
DOL tables	2K (2048 bytes)	1964	1958 bytes
	4K (4096 bytes)	4012	4006 bytes
	8K (8192 bytes)	8108	8102 bytes
	16K (16384 bytes)	16300	16294 bytes if table does not include any variable length columns
	16K (16384 bytes)	16300 (subject to a max start offset of varlen = 8191)	8191-6-2 = 8183 bytes if table includes at least one variable length column.*

* This size includes six bytes for the row overhead and two bytes for the row length field

- The maximum number of bytes of variable length data per row depends on the locking scheme for the table. Table 4-2 describes the maximum size columns for an APL table:

Table 4-2: Maximum size for variable-length columns in an APL table

Page size	Maximum row length	Maximum column length
2K (2048 bytes)	1962	1948
4K (4096 bytes)	4010	3988
8K (8192 bytes)	8096	8058
16K (16384 bytes)	16298	16228

Table 4-3 describes the maximum size of columns for a DOL table:

Table 4-3: Maximum size for variable-length columns in an DOL table

Page size	Maximum row length	Maximum column length
2K (2048 bytes)	1964	1954
4K (4096 bytes)	4012	4002
8K (8192 bytes)	8108	7998
16K (16384 bytes)	16300	162290

Table 4-3 describes the maximum size of columns for a DOL table:

Table 4-4: Maximum size for variable-length columns in an DOL table

Page size	Maximum row length	Maximum column length
2K (2048 bytes)	1964	1954
4K (4096 bytes)	4012	4002
8K (8192 bytes)	8108	7998
16K (16384 bytes)	16300	162290

- If you create a DOL table with a variable-length column that exceeds a 8191-byte offset, you cannot add any rows to the column
- If you create tables with varchar, nvarchar, univarchar, 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.

- You cannot issue create table with a declarative default or check constraint and then insert data into the table in the same batch or procedure. Either separate the create and insert statements into two different batches or procedures, or use execute to perform the actions separately.
- You cannot use the following variable in create table statements that include defaults:


```
declare @p int
select @p = 2
create table t1 (c1 int default @p, c2 int)
```

Doing so results in error message 154, which says, “Variable is not allowed in default.”

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 `money`, are reserved types that cannot be used to create columns, variables, or parameters:

Table 4-5: Variable-length datatypes used to store nulls

Original fixed-length datatype	Converted to
char	varchar
nchar	nvarchar
binary	varbinary
datetime	datetime
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 4-6: 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 <code>sp_bindrule</code>
default clause	create default and <code>sp_bindefault</code>

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 4-7: 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
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. Gaps can also occur due to transaction rollbacks, the deletion of rows, or the manual insertion of data into the IDENTITY column. The maximum size of the gap depends on the setting of the identity_burning set factor and identity_gap_size configuration parameters, the identity_gap value given in the create table or select into statement. For details about using the different methods to set the identity gap, see “Managing Identity Gaps in Tables” in Chapter 7, “Creating Databases and Tables” in the *Transact-SQL User’s Guide*.

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, but 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 4-8 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 4-8: Space management properties and locking schemes

Property	allpages	datapages	datarows
<code>max_rows_per_page</code>	Yes	No	No
<code>exp_row_size</code>	No	Yes	Yes
<code>reservepagegap</code>	Yes	Yes	Yes
<code>fillfactor</code>	Yes	Yes	Yes

- Table 4-9 shows the default values and the effects of using default values for the space management properties.

Table 4-9: Defaults and effects of space management properties

Property	Default	Effect of Using the Default
<code>max_rows_per_page</code>	0	Fits as many rows as possible on the page, up to a maximum of 255
<code>exp_row_size</code>	0	Uses the server-wide default value, set with the configuration parameter <code>default exp_row_size percent</code>
<code>reservepagegap</code>	0	Leaves no empty pages during extent allocations
<code>fillfactor</code>	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 4-10 shows when *reservepagegap* is applied.

Table 4-10: 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	Yes	Yes

(For allpages-locking to data-only locking, or vice versa)

- 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 create 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, the value stored cannot exceed 16K bytes, depending on the page size of the database server and other variables.
- If off row is specified:
 - 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 about 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

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The following are Transact-SQL extensions: <ul style="list-style-type: none"> • 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 <i>segment_name</i> 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	<p><i>Commands</i> – alter table, create existing table, create index, create rule, create schema, create view, drop index, drop rule, drop table</p> <p><i>System procedures</i> – 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</p>

create trigger

Description	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	<pre>create trigger [owner.]trigger_name on [owner.]table_name for {insert , update , delete} as SQL_statements</pre> <p>Or, using the if update clause:</p> <pre>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]...</pre>
Parameters	<p><i>trigger_name</i> 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 <i>owner</i> is the current user. If you use an owner name to qualify a trigger, you must explicitly qualify the table name the same way.</p> <p>You cannot use a variable for a trigger name.</p>

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” on page 161 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

Example 1 Prints a message when anyone tries to add data or change data in the titles table:

```
create trigger reminder
on titles
for insert, update as
print "Don't forget to print a report for
accounting."
```

Example 2 Prevents insertion of a new row into titleauthor if there is no corresponding title_id in the titles table:

```
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
```

Example 3 If the `pub_id` column of the `publishers` table is changed, make the corresponding change in the `titles` table:

```
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
```

Example 4 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:

```
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
```

Example 5 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:

```
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
```

Usage

- 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.

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.

Inserting and updating 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.

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`.

Standards

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Security Officer can grant or revoke permissions to create triggers. The Database Owner can create triggers on any user table. Users can only create triggers on tables that they own.

Permission to issue the `create trigger` command is granted to users by default.

When the System Security Officer revokes 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, issue two grant commands: the first command removes the revoke row from sysprotects; the second inserts a grant row. If permission to create triggers is revoked, 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 at trigger creation When you create a trigger, Adaptive Server makes no permission checks on objects such as tables or 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 at trigger execution 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 procedure, 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

Description

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]
```

Parameters

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” on page 190. 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” on page 190.

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

Example 1 Creates a view derived from the title, type, price and pubdate columns of the base table titles:

```
create view titles_view
as select title, type, price, pubdate
from titles
```

Example 2 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:

```
create view "new view" ("column 1", "column 2")
as select coll, col2 from "old view"
```

Example 3 Creates a view which contains the titles, advances and amounts due for books with a price less than \$5.00:

```
create view accounts (title, advance, amt_due)
as select title, advance, price * total_sales
from titles
where price > $5
```

Example 4 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:

```
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
```

Example 5 Creates a view with the same definition as in example 3, but with column headings provided in the select statement:

```
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
```

Example 6 Creates a view, `author_codes`, derived from `titleauthor` that lists the unique author identification codes:

```
create view author_codes
as select distinct au_id
from titleauthor
```

Example 7 Creates a view, `price_list`, derived from `title` that lists the unique book prices:

```
create view price_list (price)
as select distinct price
from titles
```

Example 8 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:

```
create view stores_cal
as select * from stores
where state = "CA"
with check option
```

Example 9 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_cal30` 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":

```
create view stores_cal30
as select * from stores_cal
where payterms = "Net 30"
```

Example 10 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:

```
create view stores_cal30_check
as select * from stores_cal
where payterms = "Net 30"
with check option
```

Usage

- 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 1024.
- 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.

- You cannot use the following variable in create view statements:

```
declare @p int
select @p = 2
create view v2
as
select * from t1 where c1 > @p
```

Doing so results in error message 7351, which says, “Local or global variables not allowed in view definition.”

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 returns a row for every (18 rows)—more data than you might intend:

```
select title_id, type, sum(total_sales)
from titles
group by type
```

While this query returns one row for each type (6 rows):

```
select type, sum(total_sales)
from titles
group by type
```

For more information about `group by`, see “`group by` and `having` Clauses on page 280.”

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

Standard	Compliance Level	Comments
SQL92	Entry level compliant	The use of more than one distinct keyword and the use of “ <i>column_heading = column_name</i> ” 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 at view reation 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 at view execution 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

Commands: *dbcc* – *delete statistics*

dbcc

Description

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 )
```

Parameters**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 8, “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 Chapter 37, “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

Example 1 Checks pubs2 for page allocation errors:

```
dbcc checkalloc(pubs2)
```

Example 2 Checks database consistency for pubs2 and places the information in the dbccdb database:

```
dbcc checkstorage(pubs2)
```

Example 3 Adaptive Server returns an optimized report of allocation for this table, but does not fix any allocation errors:

```
dbcc tablealloc(publishers, null, nofix)
```

```
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.
```

```
dbcc checktable(salesdetail)
```

Example 4 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:

```
dbcc indexalloc ("pubs..titleauthor", 2, full)
```

Example 5 Rebuilds or creates an internal Adaptive Server 12.x data structure for all text and image columns in the blurbs table:

```
dbcc rebuild_text (blurbs)
```

Example 6 dbcc reindex has discovered one or more corrupt indexes in the titles table:

```
dbcc reindex(titles)
```

```
One or more indexes are corrupt. They will be rebuilt.
```

Example 7 Upgrades text values for blurbs after a character set change:

```
dbcc fix_text(blurbs)
```

Example 8 Heuristically aborts the transaction, “distributedxact1”:

```
dbcc complete_xact (distributedxact1, "rollback")
```

Example 9 Removes information for the transaction, “distributedxact1” from master.dbo.systransactions:

```
dbcc forget_xact (distributedxact1)
```

Usage

- 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

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

Description	Makes a cursor inaccessible and releases all memory resources committed to that cursor.				
Syntax	<code>deallocate cursor <i>cursor_name</i></code>				
Parameters	<i>cursor_name</i> is the name of the cursor to deallocate.				
Examples	Deallocates the cursor named “authors_crsr”: <pre>deallocate cursor authors_crsr</pre>				
Usage	<ul style="list-style-type: none">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.<code>deallocate cursor</code> 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	<table border="1"><thead><tr><th>Standard</th><th>Compliance Level</th></tr></thead><tbody><tr><td>SQL92</td><td>Transact-SQL extension</td></tr></tbody></table>	Standard	Compliance Level	SQL92	Transact-SQL extension
Standard	Compliance Level				
SQL92	Transact-SQL extension				
Permissions	<code>deallocate cursor</code> permission defaults to all users. No permission is required to use it.				
See also	<i>Commands</i> – close, declare cursor				

declare

Description	Declares the name and type of local variables for a batch or procedure.
Syntax	Variable declaration: <pre>declare @<i>variable_name</i> datatype [, @<i>variable_name</i> datatype]...</pre> Variable assignment: <pre>select @<i>variable</i> = {<i>expression</i> <i>select_statement</i>} [, @<i>variable</i> = {<i>expression</i> <i>select_statement</i>} ...] [from <i>table_list</i>] [where <i>search_conditions</i>] [group by <i>group_by_list</i>]</pre>

	<pre>[having <i>search_conditions</i>] [order by <i>order_by_list</i>] [compute <i>function_list</i> [by <i>by_list</i>]]</pre>
Parameters	<p><i>@variable_name</i> must begin with @ and must conform to the rules for identifiers.</p> <p><i>datatype</i> can be either a system datatype or a user-defined datatype.</p>
Examples	<p>Example 1 Declares two variables and prints strings according to the values in the variables:</p> <pre>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</pre> <p>Example 2 Prints “Ouch!” if the maximum book price in the titles table is more than \$20.00:</p> <pre>declare @veryhigh money select @veryhigh = max(price) from titles if @veryhigh > \$20 print "Ouch!"</pre>
Usage	<ul style="list-style-type: none"> • Assign values to local variables with a select statement. • The maximum number of parameters in a procedure is 2048. 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

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

Description

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

Example 1 Defines a result set for the authors_crsr cursor that contains all authors from the authors table who do not reside in California:

```
declare authors_crsr cursor
```



```

for select au_id, au_lname, au_fname
from authors
where state != 'CA'

```

Example 2 Defines a read-only result set for the titles_crsr cursor that contains the business-type books from the titles table:

```

declare titles_crsr cursor
for select title, title_id from titles
where title_id like "BU%"
for read only

```

Example 3 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:

```

declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state

```

Usage

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.
- You can include up to 1024 columns in an update clause of a client's declare cursor statement
- *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.

Cursor scope

- 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

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

Description

Removes rows from a table.

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
```

Parameters

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

Example 1 Deletes all rows from the authors table:

```
delete authors
```

Example 2 Deletes a row or rows from the authors table:

```
delete from authors
where au_lname = "McBadden"
```

Example 3 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.):

```
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
```

Example 4 Deletes a row from the titles table currently indicated by the cursor title_crshr:

```
delete titles where current of title_crshr
```

Example 5 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:

```
delete authors
where syb_identity = 4
```

Example 6 Deletes rows from authors, skipping any locked rows:

```
delete from authors from authors readpast
where state = "CA"
```

Example 7 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:

```
delete stores from stores readpast, authors
where stores.city = authors.city
```

Usage

- 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

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. <code>readpast</code> 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

Description

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

Example 1 Delete the densities, selectivities, and histograms for all columns in the `titles` table:

```
delete statistics titles
```

Example 2 Deletes densities, selectivities, and histograms for the pub_id column in the titles table:

```
delete statistics titles(pub_id)
```

Example 3 Deletes densities, selectivities, and histograms for pub_id, pubdate, without affecting statistics on the single-column pub_id or the single-column pubdate:

```
delete statistics titles(pub_id, pubdate)
```

Usage

- 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

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

Utilities – optdiag

Commands: *disk init* – *disk unmirror*

disk init

Description	Makes a physical device or file usable by Adaptive Server.
Syntax	<pre> disk init name = "device_name" , physname = "physicalname" , [vdevno = virtual_device_number ,] size = number_of_blocks [, vstart = virtual_address , cntrltype = controller_number] [, contiguous] [, dsync = { true false }] </pre>
Parameters	<p>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.</p> <p>physname is the full specification of the database device. This name must be enclosed in single or double quotes.</p> <p>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 <code>sp_configure</code>, and check the number of devices value.</p> <p>To determine the virtual device number, look at the <code>device_number</code> column of the <code>sp_helpdevice</code> report, and use the next unused integer.</p>

size

is the amount of space to allocate to the database extension.

size can be in the following unit specifiers: 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.

vstart

is the starting virtual address, or the offset, for Adaptive Server to begin using the database device. vstart accepts the following optional unit specifiers: k or K (kilobytes), m or M (megabytes), and g or G (gigabytes). The size of the offset depends on how you enter the value for vstart.

- If you do not specify a unit size, vstart uses 2K pages for its starting address. For example, if you specify `vstart = 13`, Adaptive Server uses $13 * 2K$ pages as the offset for the starting address.
- If you specify a unit value, vstart uses this as the starting address. For example, if you specify `vstart = "13M"`, Adaptive Server sets the starting address offset at 13 megabytes.

The default value (and usually the preferred value) of vstart is 0. If the specified device does not have the sum of vstart + size blocks available, the disk init command fails. If 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.

dsync

specifies whether writes to the database device take place directly to the storage media, or are buffered when using UNIX operating system files. This option is meaningful only when you are initializing a UNIX operating system file; it has no effect when initializing devices on a raw partition. By default, all UNIX operating system files are initialized with dsync set to true.

Examples

Example 1 Initializes 5MB of a disk on a UNIX system:

```
disk init
  name = "user_disk",
  physname = "/dev/rxyla",
  vdevno = 2, size = 5120
```

Example 2 Initializes 10MB of a disk on a UNIX operating system file. Adaptive Server opens the device file with the *dsync* setting, and writes to the file are guaranteed to take place directly on the storage media:

```
disk init
  name = "user_file",
  physname = "/usr/u/sybase/data/userfile1.dat",
  vdevno = 2, size = 5120, dsync = true
```

Usage

- 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.
- You can specify the size as a float, but the size is rounded down to the nearest multiple of 2K.
- If you do not use a unit specifier for *size*:
 - *disk init* uses the virtual page size of 2K.
 - The size argument for create database and alter database is in terms of megabytes of disk space. This value is converted to the number of logical pages the master device was built with
- 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*.
- 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 `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.

Using *dsync*

Note Do not set `dsync` to false for any device that stores critical data. The only exception is `tempdb`, which can safely be stored on devices for which `dsync` is set to false.

- When `dsync` is on, writes to the database device are guaranteed to take place on 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 data updates have ever taken place on the physical media, and Adaptive Server may be unable to recover the database.
- `dsync` is always on for the master device file.

- The `dsync` value should be turned off only when 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—writes to those device are guaranteed to take place on the physical storage media, regardless of the `dsync` setting.
- The `dsync` setting is not used on the Windows NT platform.
- `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.

Standards

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

Description

Creates a software mirror that immediately takes over when the primary device fails.

Syntax

```
disk mirror
  name = "device_name" ,
  mirror = "physicalname"
  [, writes = { serial | noserial }]
```

Parameters

`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.

Examples

```
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*.

Usage

- 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, 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 starts a master device named `master.dat` and its mirror, `mirror.dat`:

```
dataserver -dmaster.dat -rmirror.dat
```

For more information, see `dataserver` and `startserver` in the *Utility Guide*.

- 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`.

Standards

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

Utilities – dataserver, startserver

disk refit

Description

Rebuilds the master database’s sysusages and sysdatabases system tables from information contained in sysdevices.

Syntax

disk refit

Examples

disk refit

Usage

- 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

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

Description	Rebuilds the master database's sysdevices system table. Use disk reinit as part of the procedure to restore the master database.
Syntax	<pre>disk reinit name = "device_name", physname = "physicalname" , [vdevno = virtual_device_number ,] size = number_of_blocks [, vstart = virtual_address , cntrltype = controller_number] [, dsync = { true false }]</pre>
Parameters	<p>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.</p> <p>physname is the name of the database device. The physical name must be enclosed in single or double quotes.</p> <p>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.</p> <p>size is the amount of space to allocate to the database extension. size can be in the following unit specifiers: 'k' or 'K' (kilobytes), 'm' or 'M' (megabytes), and 'g' or 'G' (gigabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier.</p>

vstart

is the starting virtual address, or the offset, for Adaptive Server to begin using the database device. `vstart` accepts the following optional unit specifiers: `k` or `K` (kilobytes), `m` or `M` (megabytes), and `g` or `G` (gigabytes). The size of the offset depends on how you enter the value for `vstart`.

- If you do not specify a unit size, `vstart` uses 2K pages for its starting address. For example, if you specify `vstart = 13`, Adaptive Server uses 13 * 2K pages as the offset for the starting address.
- If you specify a unit value, `vstart` uses this as the starting address. For example, if you specify `vstart = "13M"`, Adaptive Server sets the starting address offset at 13 megabytes.

The default value (and usually the preferred value) of `vstart` is 0. If the specified device does not have the sum of `vstart + size` blocks available, the `disk reinit` command fails. If 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.

dsync

specifies whether writes to the database device take place directly to the storage media, or are buffered when using UNIX operating system files. This option is meaningful only when you are initializing a UNIX operating system file; it has no effect when initializing devices on a raw partition. By default, all UNIX operating system files are initialized with `dsync` set to true.

Examples

Initializes 10MB of a disk on a UNIX operating system file. Adaptive Server opens the device file with the `dsync` setting, and writes to the file are guaranteed to take place directly on the storage media:

```
disk reinit
name = "user_file",
physname = "/usr/u/sybase/data/userfile1.dat",
vdevno = 2, size = 5120, dsync = true
```

Usage

- `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.
- You can specify the *size* as a float, but the size is rounded down to the nearest multiple of 2K.
- If you do not use a unit specifier for *size*, *disk reinit* uses the virtual page size of 2K.
- For complete information on restoring the master database, see the *System Administration Guide*.

Using *dsync*

Note Do not set *dsync* to false for any device that stores critical data. The only exception is *tempdb*, which can safely be stored on devices for which *dsync* is set to false.

- When *dsync* is on, writes to the database device are guaranteed to take place on 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 data updates have ever taken place on the physical media, and Adaptive Server may be unable to recover the database.
- *dsync* is always on for the master device file.
- The *dsync* value should be turned off only when 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—writes to those device are guaranteed to take place on the physical storage media, regardless of the *dsync* setting.
- The *dsync* setting is not used on the Windows NT platform.
- *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.

Standards

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

Description

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"
```

Parameters

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

Resumes software mirroring on the database device user_disk:

```
disk remirror  
    name = "user_disk"
```

Usage

- 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, when you restart Adaptive Server with the dataserver utility program. Add this option to the *RUN_servename* 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.

Standards

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

Utilities – dataserver, startserver

disk unmirror

Description 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 }]`

Parameters

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 `phname` 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

Example 1 Suspends software mirroring for the database device

`user_disk`:

```
disk unmirror
name = "user_disk"
```

Example 2 Suspends software mirroring for the database device `user_disk` on the secondary side:

```
disk unmirror name = "user_disk", side = secondary
```

Example 3 Suspends software mirroring for the database device `user_disk` and removes all device references to the mirror device:

```
disk unmirror name = "user_disk", mode = remove
```

Usage

- 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.
- 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

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

Utilities – dataserver, startserver

Commands: *drop database – drop view*

drop database

Description	Removes one or more databases from Adaptive Server.
Syntax	<code>drop database <i>database_name</i> [, <i>database_name</i>]...</code>
Parameters	<p><i>database_name</i></p> <ul style="list-style-type: none"> – is the name of a database to remove. Use <code>sp_helpdb</code> to get a list of databases.
Examples	<p>Example 1 Removes the publishing database:</p> <pre>drop database publishing</pre> <p>Example 2 The dropped databases (and their contents) are gone:</p> <pre>drop database publishing, newpubs</pre>
Usage	<ul style="list-style-type: none"> • Removing a database deletes the database and all its objects, frees its storage allocation, and erases its entries from the <code>sysdatabases</code> and <code>sysusages</code> system tables in the master database. • <code>drop database</code> clears the suspect page entries pertaining to the dropped database from <code>master..sysattributes</code>. <p>Restrictions</p> <ul style="list-style-type: none"> • 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 <code>drop database</code> 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: <pre>select object_name(tableid), frgndbname from sysreferences where frgndbname is not null</pre>

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

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

Description

Removes a user-defined default.

Syntax

```
drop default [owner.]default_name  
[, [owner.]default_name]...
```

Parameters

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

Removes the user-defined default datedefault from the database:

```
drop default datedefault
```

Usage

- You cannot drop a default that is currently bound to a column or to a user-defined datatype. Use the system procedure sp_unbinddefault 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

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 function (SQLJ)

Description

Removes a SQLJ function.

Syntax

```
drop func[tion] [owner.]function_name
[, [owner.]function_name ] ...
```

Parameters

[owner.]function_name
is the SQL name of a SQLJ function.

Examples

Deletes the SQLJ function square_root:

```
drop function square_root
```

Usage

- See *Java in Adaptive Server Enterprise* for more information about SQLJ functions.
- drop function removes only user-created functions from the current database. It does not remove system functions.

Permissions

Only the Database Owner or user with the sa role can execute drop function.

See also

Commands – create function (SQLJ)

drop index

Description	Removes an index from a table in the current database.				
Syntax	<code>drop index <i>table_name.index_name</i> [, <i>table_name.index_name</i>]..</code>				
Parameters	<p><i>table_name</i> is the table in which the indexed column is located. The table must be in the current database.</p> <p><i>index_name</i> 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.</p>				
Examples	<p>The index <code>au_id_ind</code> in the <code>authors</code> table no longer exists:</p> <pre>drop index authors.au_id_ind</pre>				
Usage	<ul style="list-style-type: none">• Once the <code>drop index</code> 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 <code>drop index</code> on system tables.• <code>drop index</code> cannot remove indexes that support unique constraints. To drop such indexes, drop the constraints through <code>alter table</code> or drop the table. See <code>create table</code> 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 <code>sp_cursorinfo</code>.• To get information about what indexes exist on a table, use:<pre>sp_helpindex objname</pre>where <code>objname</code> is the name of the table.				
Standards	<table border="1"><thead><tr><th>Standard</th><th>Compliance Level</th></tr></thead><tbody><tr><td>SQL92</td><td>Transact-SQL extension</td></tr></tbody></table>	Standard	Compliance Level	SQL92	Transact-SQL extension
Standard	Compliance Level				
SQL92	Transact-SQL extension				
Permissions	<code>drop index</code> permission defaults to the index owner and is not transferable.				
See also	<p><i>Commands</i> – <code>create index</code></p> <p><i>System procedures</i> – <code>sp_cursorinfo</code>, <code>sp_helpindex</code>, <code>sp_spaceused</code></p>				

drop procedure

Description	Removes a procedure.
Syntax	<code>drop proc[edure] [owner.]procedure_name [, [owner.]procedure_name] ...</code>
Parameters	<i>procedure_name</i> is the name of the Transact-SQL or SQLJ 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 <i>owner</i> is the current user.
Examples	<p>Example 1 Deletes the stored procedure showind:</p> <pre>drop procedure showind</pre> <p>Example 2 Unregisters the extended stored procedure xp_echo:</p> <pre>drop procedure xp_echo</pre>
Usage	<ul style="list-style-type: none"> • <code>drop procedure</code> 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 <code>drop procedure</code> statement. For example, if the procedures used with the application named orders were named <code>orderproc;1</code>, <code>orderproc;2</code>, and so on, the following statement: <pre>drop proc orderproc</pre> drops the entire group. Once procedures have been grouped, individual procedures within the group cannot be dropped. For example, the statement: <pre>drop procedure orderproc;2</pre> is not allowed. <p>You cannot drop extended stored procedures as a procedure group.</p> • The system procedure <code>sp_helptext</code> displays the procedure's text, which is stored in <code>syscomments</code>. • The system procedure <code>sp_helpextendedproc</code> 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

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, create procedure (SQLJ)

System procedures – sp_depends, sp_dropextendedproc, sp_helpextendedproc, sp_helptext, sp_rename

drop role

Description

Drops a user-defined role.

Syntax

drop role *role_name* [with override]

Parameters

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

Example 1 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:

```
drop role doctor_role
```

Example 2 Drops the named role and removes permission information and any other reference to the role from all databases:

```
drop role doctor_role with override
```

Usage	<ul style="list-style-type: none"> 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				
	<ul style="list-style-type: none"> You cannot use drop role to drop system roles. 				
Standards	<table border="1"> <thead> <tr> <th>Standard</th> <th>Compliance Level</th> </tr> </thead> <tbody> <tr> <td>SQL92</td> <td>Transact-SQL extension</td> </tr> </tbody> </table>	Standard	Compliance Level	SQL92	Transact-SQL extension
Standard	Compliance Level				
SQL92	Transact-SQL extension				
Permissions	<p>You must be a System Security Officer to use drop role.</p> <p>drop role permission is not included in the grant all command.</p>				
See also	<p><i>Commands</i> – alter role, create role, grant, revoke, set</p> <p><i>System procedures</i> – sp_activeroles, sp_displaylogin, sp_displayroles, sp_helprotect, sp_modifylogin</p>				

drop rule

Description	Removes a user-defined rule.
Syntax	drop rule [owner.]rule_name [, [owner.]rule_name]...
Parameters	<p><i>rule_name</i></p> <p>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 <i>owner</i> is the current user.</p>
Examples	<p>Removes the rule pubid_rule from the current database:</p> <pre>drop rule pubid_rule</pre>
Usage	<ul style="list-style-type: none"> 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

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

Description

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 ]...
```

Parameters

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

Removes the table roysched and its data and indexes from the current database:

```
drop table roysched
```

Usage

- 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 7-1: 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

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

Description

Removes a trigger.

Syntax

```
drop trigger [owner.]trigger_name  
[, [owner.]trigger_name]..
```

Parameters

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

Removes the trigger `trigger1` from the current database:

```
drop trigger trigger1
```

Usage

- `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

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

Description

Removes one or more views from the current database.

Syntax

drop view [*owner*.]*view_name* [, [*owner*.]*view_name*]...

Parameters

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

Removes the view `new_price` from the current database:

```
drop view new_price
```

Usage

- 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

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

Commands: *dump database* – *dump transaction*

dump database

Description 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 [compress::compression_level::]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]
[stripe on [compress::compression_level::]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]]
[[stripe on [compress::compression_level::]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}
} ]
```

Parameters

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.

compress::compression_level

is a number between 0 and 9, with 0 indicating no compression, and 9 providing the highest level of compression. If you do not specify *compression_level*, the default is 1. See Chapter 27, “Backing Up and Restoring User Databases” in the *System Administration Guide* for more information about the compress option.

Note The compress option works only with local archives; you cannot use the *backup_server_name* option.

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. 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. 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 `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” on page 238 for information about how to specify a dump device.

`dismount | nodismount`

on platforms 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*

on UNIX systems – when dumping to disk, 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” on page 239.

`notify = {client | operator_console}`
overrides the default message destination.

On operating systems that offer an operator terminal feature, 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

Example 1 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:

```
dump database pubs2
to "/dev/nrmt0"
```

Example 2 *For UNIX* – 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:

```
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
```

Example 3 The `init` option initializes the tape volume, overwriting any existing files:

```
dump database pubs2
to "/dev/nrmt0"
with init
```

Example 4 Rewinds the dump volumes upon completion of the dump:

```
dump database pubs2
to "/dev/nrmt0"
with unload
```

Example 5 *For UNIX* – 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:

```
dump database pubs2
to "/dev/nrmt0"
with notify = client
```

Example 6 Creates a compressed dump of the pubs2 database into a file called *dmp090100.dmp* using a compression level of 4:

```
dump database pubs2 to
"compress:4::/opt/bin/Sybase/dumps/dmp090100.dmp"
```

Usage

- Table 8-1 describes the commands and system procedures used to back up databases:

Table 8-1: 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

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).
- 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. The Backup Server must be listed in the `master..syssservers` 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_name* has different meanings depending on whether you are dumping to disk or to a UNIX tape:

`file = file_name`

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:

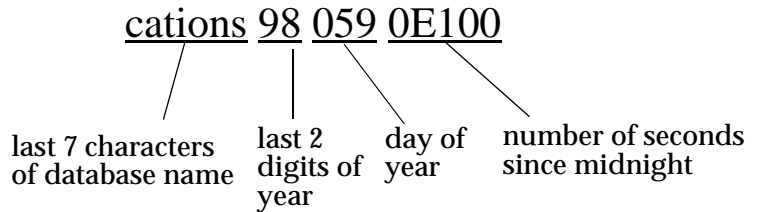
- If the dump is to tape, 'filename' is the name of the tape device;
- If the dump is to disk, it is the name of a disk file.

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 *publications980590E100* contains a copy of the publications database made on the fifty-ninth day of 1998:

Figure 8-1: File naming convention for database dumps to tape



- 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.

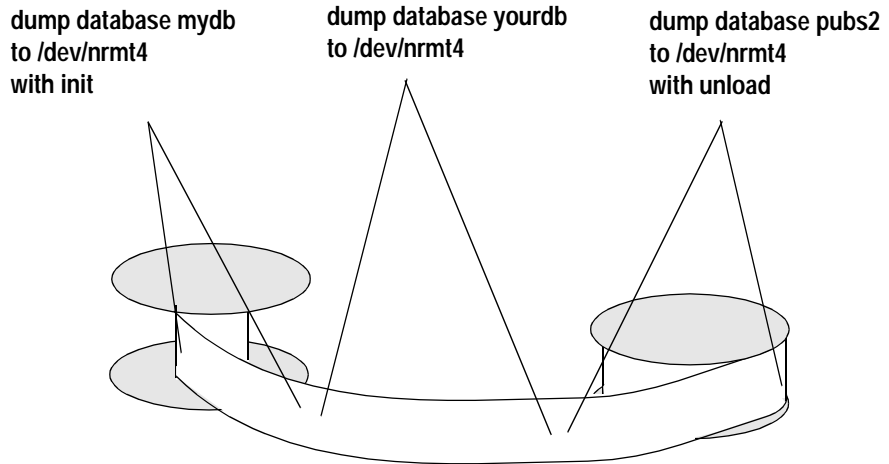
Changing dump volumes

- *On 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 8-2 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 8-2: Dumping several databases to the same volume



Dumping from a 32-bit OS to a 64-bit OS

Database dumps from a 32-bit version of Adaptive Server are fully compatible with a 64-bit version of Adaptive Server of the same platform, and vice-versa.

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

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_helppdb, sp_helpthreshold, sp_logdevice, sp_spaceused, sp_volchanged

dump transaction

Description 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 [compress::[compression_level::]]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]
[stripe on [compress::[compression_level::]]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]]
[[stripe on [compress::[compression_level::]]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 [compress::[compression_level::]]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]
[[stripe on [compress::[compression_level::]]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
capacity = number_kilobytes,
dumpvolume = volume_name,
file = file_name]]
[[[stripe on [compress::[compression_level::]]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}}
```

Parameters

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.

`compress::compression_level`

is a number between 0 and 9, with 0 indicating no compression, and 9 providing the highest level of compression. If you do not specify `compression_level`, the default is 1. See Chapter 27, “Backing Up and Restoring User Databases” in the *System Administration Guide* for more information about the `compress` option.

Note The `compress` option works only with local archives; you cannot use the `backup_server_name` option.

`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” on page 238 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. 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.

`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.

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 `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” on page 238 for information about how to specify a dump device.

dismount | nodismount

on platforms 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*

on UNIX – 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 multfile 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” on page 239.

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 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

Example 1 Dumps the transaction log to a tape, appending it to the files on the tape, since the `init` option is not specified:

```
dump transaction pubs2
to "/dev/nrmt0"
```

Example 2 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. The `retaindays` option specifies that the tapes cannot be overwritten for 14 days:

```
dump transaction mydb
to "/dev/nrmt4" at REMOTE_BKP_SERVER
stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
with init, retaindays = 14
```

Example 3 Dumps completed transactions from the `inventory_db` transaction log file to device `dev1`:

```
dump tran inventory_db to dev1 with standby_access
```

- Usage
- Table 8-2 describes the commands and system procedures used to back up databases and logs.

Table 8-2: 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).
- 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/pilsort 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

- 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.

Dumping databases without separate log segments

- 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 `truncate_only` on the master, model, and `sysystemprocs` 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.

Dumping only complete transactions

- 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.

Dumping without the log

Warning! 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` provides no means to recover your databases. Run `dump database` at the earliest opportunity to ensure recoverability.

- `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.
- 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. The Backup Server must be listed in the master..syssservers 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.

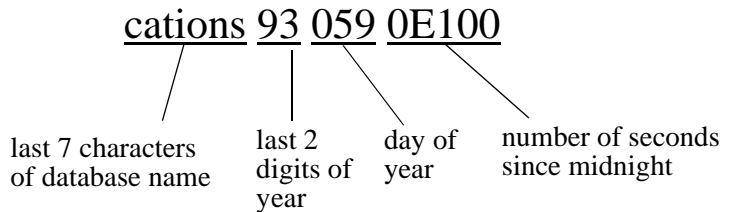
Dump files

- 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 8-3: 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.

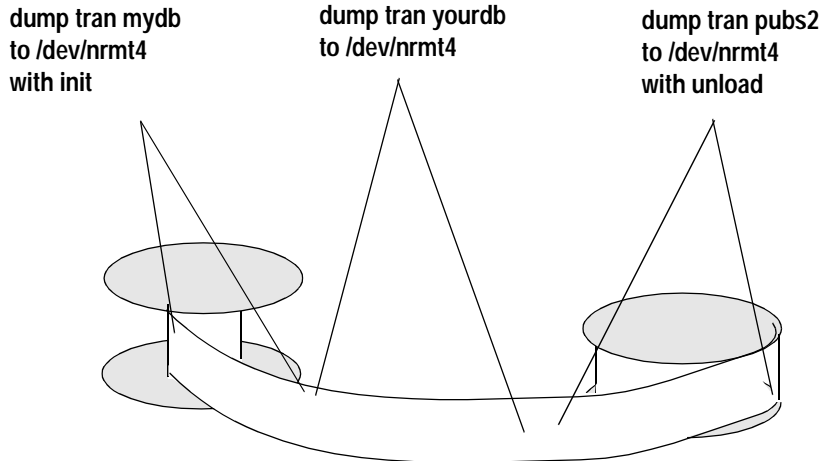
Note When dumping and loading across the network, you must specify the same number of stripe devices for each operation.

Changing dump volumes

- *On 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 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 8-4 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 8-4: 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

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

Commands: *execute* – *group by having* clauses

execute

Description	Runs a procedure or dynamically executes Transact-SQL commands
Syntax	<pre> [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]...) </pre>
Parameters	<p>execute exec</p> <p>is used to execute a stored procedure or an extended stored procedure (ESP). It is necessary only if the stored procedure call is <i>not</i> the first statement in a batch.</p> <p>@return_status</p> <p>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.</p> <p>server</p> <p>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.</p> <p>database</p> <p>is the database name. Specify the database name if the procedure is in another database. The default value for <i>database</i> 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.</p>

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.

Examples

Example 1 All three examples execute the stored procedure showind with a parameter value titles:

```
execute showind titles
```

or:

```
exec showind @tabname = titles
```

or, if this is the only statement in a batch or file:

```
showind titles
```

Example 2 Executes the stored procedure checkcontract on the remote server GATEWAY. Stores the return status indicating success or failure in *@retstat*:

```
declare @retstat int
execute @retstat = GATEWAY.pubs.dbo.checkcontract
"409-56-4008"
```

Example 3 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*:

```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent
output
select Percent = @percent
```

Example 4 This procedure displays information about the system tables if you do not supply a parameter:

```
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
```

Example 5 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*:

```
declare @input varchar(12), @in varchar(12),
        @out varchar(255), @result varchar(255)
select @input="Hello World!"
execute xp_echo @in = @input, @out= @result output
```

Example 6 The final `execute` command concatenates string values and character variables to issue the Transact-SQL command:

```
select name from sysobjects where id=3

declare @tablename char(20)
declare @columnname char(20)
select @tablename="sysobjects"
select @columnname="name"
execute ('select ' + @columnname + ' from ' +
@tablename + ' where id=3')
```

Example 7 Executes the stored procedure `sp_who`:

```
declare @sproc varchar(255)
select @sproc = "sp_who"
execute @sproc
```

Usage

- 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
-----
```

```
sybssystemprocs
(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.

- The Dynamic SQL syntax of `exec (@parameter_name)` is also valid; however, it may take more keystrokes. For example, the Dynamic SQL command `exec (@sproc = "7")` passes the integer value 7 to the procedure, but this can be accomplished with fewer keystrokes as `exec @sproc 7`.
- 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, connect to, 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

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

Description

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

Example 1 Returns a row of information from the cursor result set defined by the `authors_crsr` cursor:

```
fetch authors_crsr
```

Example 2 Returns a row of information from the cursor result set defined by the `pubs_crsr` cursor into the variables `@name`, `@city`, and `@state`:

```
fetch pubs_crsr into @name, @city, @state
```

Usage

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 the number of rows 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 9-1.

Table 9-1: @@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

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

Description

Branches to a user-defined label.

Syntax

label;
goto *label*

Examples

Shows the use of a label called restart:

```

declare @count smallint
select @count = 1
restart:
    print "yes"
select @count = @count + 1
while @count <=4
    goto restart

```

Usage

- 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

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

Description

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...]

```


Parameters

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 <i>permission_list</i> or <i>column_list</i> (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

Example 1 Grants Mary and the “sales” group permission to use the insert and delete commands on the titles table:

```
grant insert, delete
on titles
to mary, sales
```

Example 2 Two ways to grant update permission on the price and advance columns of the titles table to “public” (which includes all users):

```
grant update
on titles (price, advance)
to public
```

or:

```
grant update (price, advance)
on titles
to public
```

Example 3 Grants Harry and Billy permission to execute either set proxy or set session authorization to impersonate another user in the server:

```
grant set proxy to harry, billy
```

Example 4 Grants users with sso_role permission to execute either set proxy or set session authorization to impersonate another user in the server:

```
grant set session authorization to sso_role
```

Example 5 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:

```
grant set proxy to vip_role
```

Example 6 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:

```
grant create database, create table
to mary, john
```

Example 7 Grants complete access permissions on the `titles` table to all users:

```
grant all on titles
to public
```

Example 8 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:

```
grant all
to public
```

Example 9 Gives Mary permission to use the `update` command on the `authors` table and to grant that permission to others:

```
grant update on authors
to mary
with grant option
```

Example 10 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:

```
grant select, update on titles(price)
to bob
with grant option
```

Example 11 Grants permission to execute the `new_sproc` stored procedure to all System Security Officers:

```
grant execute on new_sproc
to sso_role
```

Example 12 Grants James permission to create a referential integrity constraint on another table that refers to the `price` column of the `titles` table:

```
grant references on titles(price)
to james
```

Example 13 Grants the role “specialist”, with all its permissions and privileges, to the role “doctor”:

```
grant role specialist_role to doctor_role
```

Example 14 Grants the role “doctor” to Mary:

```
grant role doctor_role to mary
```

Usage

- You can substitute the word from for to in the grant syntax.
- Table 9-2 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 9-2 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 9-2: 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			•				•	

Statement	Defaults to					Can be granted/revoked		
	System Admin	operator	Database Owner	Object owner	Public	Yes	No	N/A
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	•							
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					•			•

Statement	Defaults to					Can be granted/revoked		
	System Admin	operator	Database Owner	Object owner	Public	Yes	No	N/A
save transaction					•			•
select				• (3)		•		
set					•			•
setuser			•					•
shutdown	•							•
truncate table				•				•
update				• (3)		•		
update all statistics				•				•
update partition statistics				•				•
update statistics				•				•
writetext				•			(6)	

- 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 grants 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

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 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 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 Only a System Security Officer can grant set proxy or set session authorization, and only from the master database.

Roles 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

Description	Used in select statements to divide a table into groups and to return only groups that match conditions in the having clause.
Syntax	<i>Start of select statement</i> [group by [all] <i>aggregate_free_expression</i> [, <i>aggregate_free_expression</i>]...] [having <i>search_conditions</i>] <i>End of select statement</i>

Parameters

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.”

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

Example 1 Calculates the average advance and the sum of the sales for each type of book:

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

Example 2 Groups the results by type, then by pub_id within each type:

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

Example 3 Calculates results for all groups, but displays only groups whose type begins with “p”:

```
select type, avg(price)
from titles
group by type
having type like 'p%'
```

Example 4 Calculates results for all groups, but displays results for groups matching the multiple conditions in the having clause:

```
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"

```

Example 5 Calculates the total sales for each group (publisher) after joining the titles and publishers tables:

```

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

```

Example 6 Displays the titles that have an advance of more than \$1000 and a price that is more than the average price of all titles:

```

select title_id, advance, price
from titles
where advance > 1000
having price > avg(price)

```

Usage

- 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`” on page 285).
- 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.” 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 “Parameters” 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:

a An example of a standard grouping query:

```
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)

b 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 a):

```
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)

- c 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.

```
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)

- d 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.

```
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)

- e 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:

```
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)

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.

- f 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:

```
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)

- g 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.

```
select pub_id, count(pub_id)
from publishers
```

```
pub_id
-----
0736      3
0877      3
1389      3
```

(3 rows affected)

- h 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:

```
select pub_id, count(pub_id)
from publishers
where pub_id < "1000"
```

```
pub_id
-----
0736      2
0877      2
1389      2
```

(3 rows affected)

i 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.

```
select pub_id, count(pub_id)
from publishers
having pub_id < "1000"
```

```
pub_id
-----
0736          3
0877          3
```

(2 rows affected)

j 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:

```
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)

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

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

Description	Imposes conditions on the execution of a SQL statement.
Syntax	<pre>if <i>logical_expression</i> [plan "<i>abstract plan</i>"] <i>statements</i> [else [if <i>logical_expression</i>] [plan "<i>abstract plan</i>"] <i>statement</i>]</pre>
Parameters	<p><i>logical_expression</i> 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.</p> <p>plan "<i>abstract plan</i>" 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.</p> <p><i>statements</i> is either a single SQL statement or a block of statements delimited by begin and end.</p> <p>plan "<i>abstract plan</i>" 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 30, "Creating and Using Abstract Plans," in the <i>Performance and Tuning Guide</i>.</p>
Examples	<p>Example 1 Prints "yes" if 3 is larger than 2:</p> <pre>if 3 > 2 print "yes"</pre> <p>Example 2 The if...else condition tests for the presence of authors whose postal codes are 94705, then prints "Berkeley author" for the resulting set:</p>

```
if exists (select postalcode from authors
where postalcode = "94705")
print "Berkeley author"
```

Example 3 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:

```
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
```

Example 4 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 “Expressions, Identifiers, and Wildcard Characters.”

```
if (select total_sales
from titles
where title_id = "PC9999") > 100
select "true"
else
select "false"
```

Usage

- 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

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

Description

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" ] }
```

Parameters

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” on page 297 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 30, “Creating and Using Abstract Plans,” in the *Performance and Tuning Guide* for more information.

Examples

Example 1

```
insert titles
values("BU2222", "Faster!", "business", "1389",
       null, null, null, "ok", "06/17/87", 0)
```

Example 2

```
insert titles
(title_id, title, type, pub_id, notes, pubdate,
 contract)
```

```
values ('BU1237', 'Get Going!', 'business',
       '1389', 'great', '06/18/86', 1)
```

Example 3

```
insert newauthors
select *
from authors
where city = "San Francisco"
```

Example 4

```
insert test
select *
from test
where city = "San Francisco"
```

Usage

- 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` and `univarchar` 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`, `unichar`, `univarchar`, `varchar`, or `nvarchar` column are silently truncated unless the `string_truncation` 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 a text page 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
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

Standard	Compliance Level	Comments
SQL92	Entry level compliant	<p>The following are Transact-SQL extensions:</p> <ul style="list-style-type: none"> 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 <hr/> <p>Note The FIPS flagger does not detect insertions through a view that contains a join.</p>

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

Utilities – bcp

kill

Description

Kills a process.

Syntax

kill *spid*

Parameters

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

kill 1378

Usage

- 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 10-1: 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.
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 *spids* of the processes holding them, use sp_lock.

Standards

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

Commands: *load database – lock table*

load database

Description Loads a backup copy of a user database, including its transaction log, that was created with dump database.

Syntax

```
load database database_name
  from [compress::]stripe_device
    [at backup_server_name ]
    [density = density_value,
     blocksize = number_bytes,
     dumpvolume = volume_name,
     file = file_name]
  [stripe on [compress::]stripe_device
    [at backup_server_name ]
    [density = density_value,
     blocksize = number_bytes,
     dumpvolume = volume_name,
     file = file_name]
  [[stripe on [compress::]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}
  }]]
```

Parameters

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.

`compress::`

invokes the decompression of the archived database. See Chapter 27, “Backing Up and Restoring User Databases” in the *System Administration Guide* for more information about the `compress` option.

`from stripe_device`

is the device from which data is being loaded. See “Specifying dump devices” on page 322 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. 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.

Note When using `load database`, the `dumpvolume` option does not provide an error messages if an incorrect file name is given for the `file=filename` option. The backup server searches the entire tape looking for that file, regardless of an incorrect tape mounted.

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” on page 322 for information about how to specify a dump device.

dismount | nodismount

on platforms that support logical dismount – 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 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

Example 1 Reloads the database pubs2 from a tape device:

```
load database pubs2
  from "/dev/nrmt0"
```

Example 2 Loads the pubs2 database, using the Backup Server REMOTE_BKP_SERVER. This command names three devices:

```
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
```

Example 3 Loads the pubs2 database from a compressed dump file called *dmp090100.dmp* located at */opt/bin/Sybase/dumps*:

Usage

```
load database pubs2 from
    "compress::/opt/bin/Sybase/dumps/dmp090100.dmp"
```

- The listonly and headeronly options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
- Table 11-1 describes the commands and system procedures used to restore databases from backups:

Table 11-1: 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.
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.

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 brought online:
 - All unused pages in the database must be zeroed after the load completes. The time required depends on the number of unused pages. If the target database is the same size as the database that is loaded, the Backup Server performs this step. If the target database is larger than the database that is loaded, Adaptive Server performs this step after the Backup Server completes the load. The time required for this step depends on the number of empty 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. This step is performed by Adaptive Server.
- 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).
- 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:
 - a Load the most recent database dump.
 - b 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.

- c Issue the online database command to do the upgrade and make the database available for public use.
- d 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. The Backup Server must be listed in the `master..syssservers` 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.

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

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	<i>Commands</i> – dbcc, dump database, dump transaction, load transaction, online database <i>System procedures</i> – sp_helpdevice, sp_volchanged, sp_helpdb

load transaction

Description	Loads a backup copy of the transaction log that was created with the dump transaction command.
Syntax	<pre>load tran[saction] <i>database_name</i> from [compress::]<i>stripe_device</i> [at <i>backup_server_name</i>] [density = <i>density_value</i>, blocksize = <i>number_bytes</i>, dumpvolume = <i>volume_name</i>, file = <i>file_name</i>] [stripe on [compress::]<i>stripe_device</i> [at <i>backup_server_name</i>] [density = <i>density_value</i>, blocksize = <i>number_bytes</i>, dumpvolume = <i>volume_name</i>, file = <i>file_name</i>] [[stripe on [compress::]<i>stripe_device</i> [at <i>backup_server_name</i>] [density = <i>density_value</i>, blocksize = <i>number_bytes</i>, dumpvolume = <i>volume_name</i>, file = <i>file_name</i>]]...] [with { density = <i>density_value</i>, blocksize = <i>number_bytes</i>, dumpvolume = <i>volume_name</i>, file = <i>file_name</i>, [dismount nodismount], [nounload unload], listonly [= full], headeronly, notify = {client operator_console} until_time = <i>datetime</i>}]</pre>

Parameters

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.

compress::

invokes the decompression of the archived transaction log. See Chapter 27, “Backing Up and Restoring User Databases” in the *System Administration Guide* for more information about the compress option.

from *stripe_device*

is the name of the dump device from which you are loading the transaction log. See “Specifying dump devices” on page 322 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. 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” on page 322 for information about how to specify a dump device.

dismount | nodismount

on platforms that support logical dismount – 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 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

Example 1 Loads the transaction log for the database pubs2 tape:

```
load transaction pubs2
from "/dev/nrmt0"
```

Example 2 Loads the transaction log for the pubs2 database, using the Backup Server REMOTE_BKP_SERVER:

```
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
```

Example 3 Loads the transaction log for pubs2, up to March 20, 1997, at 10:51:43:866 a.m.:

```
load transaction pubs2
from "/dev/ntmt0"
with until_time = "mar 20, 1997 10:51:43:866am"
```

Usage

- The listonly and headeronly options display information about the dump files without loading them.
- Dumps and loads are performed through Backup Server.
- Table 11-2 describes the commands and system procedures used to restore databases from backups:

Table 11-2: 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.

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).

- 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:
 - a Load the most recent database dump.
 - b Load, *in order*, all transaction logs generated after the last database dump.
 - c Use online database to do the upgrade.
 - d 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. The Backup Server must be listed in the `master..syssservers` 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.

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

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

Description

Explicitly locks a table within a transaction.

Syntax

lock table *table_name* in {share | exclusive } mode
 [wait [*numsecs*] | nowait]

Parameters

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

Example 1 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:

```
begin transaction
lock table titles in share mode
```

Example 2 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:

```
begin transaction
lock table authors in exclusive mode wait 5
```

Example 3 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:

```
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
```

Usage

- 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

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

Description	Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a case expression.
Syntax	<code>nullif(<i>expression</i>, <i>expression</i>)</code>
Parameters	<p><code>nullif</code></p> <p>compares the values of the two expressions. If the first expression equals the second expression, <code>nullif</code> returns NULL. If the first expression does not equal the second expression, <code>nullif</code> returns the first expression.</p> <p><i>expression</i></p> <p>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” on page 179.</p>
Examples	<p>Example 1 Selects the <i>titles</i> and <i>type</i> from the <i>titles</i> table. If the book type is UNDECIDED, <code>nullif</code> returns a NULL value:</p> <pre>select title, nullif(type, "UNDECIDED") from titles</pre> <p>Example 2 This is an alternative way of writing example 1:</p> <pre>select title, case when type = "UNDECIDED" then NULL else type end from titles</pre>
Usage	<ul style="list-style-type: none"> • <code>nullif</code> expression alternate for a case expression. • <code>nullif</code> expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a when...then construct. • <code>nullif</code> 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 “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

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

Description

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

Example 1 Makes the *pubs2* database available for public use after a load sequence completes:

```
online database pubs2
```

Example 2 Brings the database inventory_db online. Used after loading inventory_db with a transaction-log dump obtained through dump tran...with standby_access:

```
online database inventory_db for standby_access
```

Usage

- 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

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

Description

Opens a cursor for processing.

Syntax

open *cursor_name*

Parameters

cursor_name

– is the name of the cursor to open.

Examples

Opens the cursor named authors_crshr:

```
open authors_crshr
```

Usage

- 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.

Standards

Standard	Compliance Level
SQL92	Entry level compliant

Permissions

open permission defaults to all users.

See also

Commands – close, declare cursor, fetch

order by Clause

Description

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]
```

Parameters

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

Example 1 Selects the titles whose price is greater than \$19.99 and lists them with the titles in alphabetical order:

```
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
```

Example 2 Lists the books from the titles table, in descending alphabetical order of the type, and calculates the average price and advance for each type:

```
select type, price, advance  
from titles  
order by type desc  
compute avg(price), avg(advance) by type
```

Example 3 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:

```
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
```


Example 4 Lists book titles and types in order by the type, renaming the columns in the output:

```
select title as BookName, type as Type
from titles
order by Type
```

Usage

- 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 12-1: 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

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

Description

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]

Usage

- For more information, see the .

Standards

Standard	Compliance Level
SQL92	Transact-SQL extension

See also

Commands – begin transaction, begin transaction, rollback, save transaction

print

Description

Prints a user-defined message on the user's screen.

Syntax

```
print
    {format_string | @local_variable |
    @ @global_variable}
    [, arg_list]
```

Parameters

format_string

can be either a variable or a string of characters. The maximum length of *format_string* is 1023 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 zulässig.
```

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

Example 1 Prints “Berkeley author” if any authors in the authors table live in the 94705 ZIP code:

```
if exists (select postalcode from authors
where postalcode = '94705')
print "Berkeley author"
```

Example 2 Declares a variable, assigns a value to the variable, and prints the value:

```
declare @msg char(50)
select @msg = "What's up, doc?"
print @msg

What's up, doc?
```

Example 3 Demonstrates the use of variables and placeholders in messages:

```
declare @tablename varchar(30)
select @tablename = "titles"

declare @username varchar(30)
select @username = "ezekiel"

print "The table '%1!' is not owned by the user
'%2!'.", @tablename, @username

The table 'titles' is not owned
by the user 'ezekiel.'
```

Usage

- The maximum output string length of *format_string* plus all arguments after substitution is 1024 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 %! 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

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

Description

Suspends and resumes updates to a specified list of databases.

Syntax

quiesce database *tag_name* hold *dbname* [, *dbname*] ...
[for external dump

	or:
	<code>quiesce database <i>tag_name</i> release</code>
Parameters	<p><i>tag_name</i> is a user-defined name that designates the list of databases to hold or release. The <i>tag_name</i> must conform to the rules for identifiers.</p> <p><i>database</i> is a database name.</p> <p><i>for external dump</i> specifies that while updates to the databases in the list are suspended, you will physically copy all affected database devices, using some facility external to Adaptive Server. The copy operation is to serve as a replacement for the combination of dump database and load database.</p>
Examples	<p>Example 1 Suspends update activity on salesdb and ordersdb:</p> <pre>quiesce database report_dbs hold salesdb, ordersdb</pre> <p>Example 2 Resumes update activity on the databases labeled report_dbs:</p> <pre>quiesce database report_dbs release</pre> <p>Example 3 Suspends update activity to the pubs2 database and signifies your intent to make an external copy of this database:</p> <pre>quiesce database pubs_tag hold pubs2 for external dump</pre>
Usage	<ul style="list-style-type: none"> • <code>quiesce database</code> used with the <code>hold</code> 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. • <code>quiesce database</code> used with the <code>release</code> keyword allows updates to resume on databases that were previously suspended. • <code>quiesce database</code> used with the <code>for external dump</code> clause signifies that you intend to make an external copy of the database. It serves to replace a combination of <code>dump</code> and <code>load database</code>. • The <code>quiesce database hold</code> and <code>release</code> 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.
- 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

Description

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]
```

Parameters

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 1024 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 zulässig.
```

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

Example 1 This stored procedure example returns an error if it does not find the table supplied with the @*tablename* parameter:

```
create procedure showtable_sp @tablename varchar(18)
as
if not exists (select name from sysobjects
where name = @tablename)
begin
raiserror 99999 "Table %1! not found.",
@tablename
end
else
```

```
begin
    select sysobjects.name, type, crdate, indid
    from sysindexes, sysobjects
    where sysobjects.name = @tabname
    and sysobjects.id = sysindexes.id
end
```

Example 2 This example adds a message to sysusermessages, then tests the message with raiserror, providing the substitution arguments:

```
sp_addmessage 25001,
    "There is already a remote user named '%1!'
    for remote server '%2!'."

raiserror 25001, jane, myserver
```

Example 3 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:

```
raiserror 20100 "Login must be at least 5
    characters long" with errordata "column" =
    "login", "server" = @@servername
```

Usage

- 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 non-fatal error.
- The maximum output string length of *format_string* plus all arguments after substitution is 1024 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

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

Description

Reads text and image values, starting from a specified offset and reading a specified number of bytes or characters.

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 ] } ]
```

Parameters

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

Example 1 Selects the second through the sixth character of the copy column:

```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
where au_id = "648-92-1872"
readtext blurbs.copy @val 1 5 using chars
```

Example 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
```

Usage

- 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

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

System procedures – text and image datatypes

reconfigure

Description

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

Usage

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

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

Description	<p>Removes one or more Java-SQL classes, packages, or JARs from a database.</p> <p>Use when Java classes are installed in the database. Refer to <i>Java in Adaptive Server Enterprise</i> for more information.</p>
Syntax	<pre>remove java class <i>class_name</i> [, <i>class_name</i>]... package <i>package_name</i> [, <i>package_name</i>]... jar <i>jar_name</i> [, <i>jar_name</i>]...[retain classes]</pre>
Parameters	<p>class <i>class_name</i> the name of one or more Java classes to be removed from the database. The classes must be installed in the current database.</p> <p>package <i>package_name</i> the name of one or more Java packages to be removed. The packages must be stored in the current database.</p> <p>jar <i>jar_name</i> either a SQL identifier or character string value of up to 30 bytes that contains a valid SQL identifier.</p> <p>Each <i>jar_name</i> must be equal to the name of a retained jar in the current database.</p> <p>retain classes specifies that the named JARs are no longer retained in the database, and the retained classes have no associated JAR.</p>
Usage	<ul style="list-style-type: none">• 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<ul style="list-style-type: none">• 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.
- You cannot remove a Java-SQL class if that class is directly referenced by a SQLJ stored procedure or function.
- To remove a Java-SQL class from the database, you must:
 - a Delete all SQLJ stored procedures or functions that directly reference the class using `drop procedure` and/or `drop function`.
 - b Delete the Java-SQL class from the database using `remove java`.

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

Description

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

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

Example 1 Reclaims unused page space in the titles table:

```
reorg reclaim_space titles
```

Example 2 Reclaims unused page space in the index titleind:

```
reorg reclaim_space titles titleind
```

Example 3 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:

```
reorg compact titles with time = 120
```

Example 4 Initiates reorg compact at the point where the previous reorg compact stopped and continues for 30 minutes:

```
reorg compact titles with resume, time = 30
```

Usage

- 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

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

Description

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"]
```

Parameters

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 30, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples

Example 1 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:

```
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
```

Example 2 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:

```
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'

```

Example 3 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):

```

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

```

Usage

- 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 12-2: 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

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

Description	Revokes permissions or roles from users or roles.
Syntax	<p>To revoke permission to access database objects:</p> <pre>revoke [grant option for] {all [privileges] <i>permission_list</i>} on { <i>table_name</i> [(<i>column_list</i>)] <i>view_name</i> [(<i>column_list</i>)] <i>stored_procedure_name</i>} from {public <i>name_list</i> <i>role_name</i>} [<i>cascade</i>]</pre> <p>To revoke permission to create database objects, execute set proxy, or execute set session authorization:</p> <pre>revoke {all [privileges] <i>command_list</i>} from {public <i>name_list</i> <i>role_name</i>}</pre> <p>To revoke a role from a user or another role:</p> <pre>revoke role {<i>role_name</i> [, <i>role_name</i> ...]} from {<i>grantee</i> [, <i>grantee</i> ...]}</pre>
Parameters	<p>all</p> <p>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.</p> <p>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.</p> <p>all does not apply to set proxy or set session authorization.</p>

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 <i>permission_list</i> or <i>column_list</i> (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

Example 1 Revokes insert and delete permissions on the titles table from Mary and the “sales” group:

```
revoke insert, delete
on titles
from mary, sales
```

Example 2 Two ways to revoke update permission on the price and advance columns of the titles table from “public”:

```
revoke update
on titles (price, advance)
from public
```

or:

```
revoke update (price, advance)
on titles
from public
```

Example 3 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:

```
revoke create database, create table
from mary, john
```

Example 4 Revokes permission from Harry and Billy to execute either set proxy or set session authorization to impersonate another user in the server:

```
revoke set proxy from harry, billy
```

Example 5 Revokes permission from users with sso_role to execute either set proxy or set session authorization:

```
revoke set session authorization from sso_role
```

Example 6 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:

```
revoke set proxy from vip_role
```

Example 7 Revokes all object creation permissions from Mary in the current database:

```
revoke all
from mary
```

Example 8 Revokes all object access permissions on the titles table from Mary:

```
revoke all
on titles
from mary
```

Example 9 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:

```
revoke references
on titles (price, advance)
from tom
```

or:

```
revoke references (price, advance)
on titles
from tom
```

Example 10 Revokes permission to execute the stored procedure new_sproc from all users who have been granted the Operator role:

```
revoke execute on new_sproc
from oper_role
```

Example 11 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:

```
revoke grant option for
insert, update, delete
on authors
from john
cascade
```

Example 12 Revokes doctor_role from specialist_role:

```
revoke role doctor_role from specialist_role
```

Example 13 Revokes “doctor_role” and “surgeon_role” from “specialist_role” and “intern_role”, and from users Mary and Tom:

```
revoke role doctor_role, surgeon_role from
specialist_role, intern_role, mary, tom
```

Usage

- 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

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Database object access 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 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 Only a System Security Officer can revoke set proxy or set session authorization, and only from the master database.

Roles 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`

Commands: *rollback* – *save transaction*

rollback

Description	Rolls back a user-defined transaction to the named savepoint in the transaction or to the beginning of the transaction.
Syntax	<code>rollback [tran transaction work] [<i>transaction_name</i> <i>savepoint_name</i>]</code>
Parameters	<code>tran transaction work</code> specifies that you want to rollback the transaction or the work. If you specify <code>tran</code> , <code>transaction</code> , or <code>work</code> , you can also specify the <i>transaction_name</i> or the <i>savepoint_name</i> . <i>transaction_name</i> is the name assigned to the outermost transaction. It must conform to the rules for identifiers. <i>savepoint_name</i> is the name assigned to the savepoint in the save transaction statement. The name must conform to the rules for identifiers.
Examples	Rolls back the transaction: <pre>begin transaction delete from publishers where pub_id = "9906" rollback transaction</pre>
Usage	<ul style="list-style-type: none">• <code>rollback transaction</code> without a <i>transaction_name</i> or <i>savepoint_name</i> rolls back a user-defined transaction to the beginning of the outermost transaction.• <code>rollback transaction <i>transaction_name</i></code> 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.• <code>rollback transaction <i>savepoint_name</i></code> rolls a user-defined transaction back to the matching save transaction <i>savepoint_name</i>.

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

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

Description 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*]

Parameters 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 Rolls back a trigger and issues the user-defined error message 25002:

```
rollback trigger with raiserror 25002
    "title_id does not exist in titles table."
```

Usage

- 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

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

Description

Sets a savepoint within a transaction.

Syntax

save transaction *savepoint_name*

Parameters

savepoint_name

is the name assigned to the savepoint. It must conform to the rules for identifiers.

Examples

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:

```
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) * royaltypcr
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

rollback transaction percentchanged

commit transaction

```

Usage

- 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 `transaction_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

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

`save transaction` permission defaults to “public.” No permission is required to use it.

save transaction

See also

Commands – begin transaction, commit, rollback

select

Description

Retrieves rows from database objects.

Syntax

```
select ::=
  select [ all | distinct ] select_list
    [ into_clause ]
    [ from_clause ]
    [ where_clause ]
    [ group_by_clause ]
    [ having_clause ]
    [ order_by_clause ]
    [ compute_clause ]
    [ read_only_clause ]
    [ isolation_clause ]
    [ browse_clause ]
    [ plan_clause ]
```

select_list ::=

Note For details on *select_list*, see the parameters description.

into_clause ::=

```
into [[database.]owner.]table_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
[existing table table_name]
[[external type] at "path_name"
[column delimiter delimiter]]
```

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"
```

Parameters

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.

existing table *table_name*

indicates that you are selecting data into a proxy table. You cannot use this select into any other table type except proxy. The column list in the select list must match the type, length, and number in the proxy table.

at "*path_name*"

indicates the full path name of the external file you are selecting into. You can only use the at parameter to select into a proxy table.

external [table | file]

indicates that the type of the external object is either a file or a table. If you do indicate either a file or a table, *select into* assumes that you are using a table.

column delimiter “*delimiter*”

indicates the delimiter that you are using to separate columns. If you do not specify a delimiter, *select into* uses the tab character.

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.

with identity_gap

specifies the identity gap for the table. This value overrides the system identity gap setting for this table only.

value

is the identity gap amount.

If you are creating a table in a *select into* statement from a table that has a specific identity gap setting, the new table does not inherit the identity gap setting from the parent table. Instead, the new table uses the identity gap setting from the parent table. To give the new table a specific *identity_gap* setting, specify the identity gap in the *select into* statement. You can give the new table an identity gap that is the same as or different from the parent table.

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_crshr 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 on page 451 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 14-1 with group by (*expression* is almost always a column name):

Table 14-1: Results of using aggregates with group by

Aggregate Function	Result
sum([all distinct] <i>expression</i>)	Total of the values in the numeric column.
avg([all distinct] <i>expression</i>)	Average of the values in the numeric column.
count([all distinct] <i>expression</i>)	Number of (distinct) non-null values in the column.

Aggregate Function	Result
count(*)	Number of selected rows.
max(<i>expression</i>)	Highest value in the column.
min(<i>expression</i>)	Lowest value in the column.

See group by and having Clauses on page 280 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 30, "Creating and Using Abstract Plans," in the *Performance and Tuning Guide* for more information.

Examples**Example 1** Selects all rows and columns from the publishers table:

```
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

Example 2 Selects all rows from specific columns of the publishers table:

```
select pub_id, pub_name, city, state from publishers
```

Example 3 Selects all rows from specific columns of the publishers table, substituting one column name and adding a string to the output:

```
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

Example 4 Selects all rows from specific columns of the titles table, substituting column names:

```
select type as Type, price as Price
from titles
```

Example 5 Specifies the locking scheme and the reserve page gap for select into:

```
select title_id, title, price
into bus_titles
lock datarows with reservepagegap = 10
from titles
where type = "business"
```

Example 6 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:

```
select title, price
from titles readpast
where type = "news"
and price between $20 and $30
```

Example 7 Selects specific columns and rows, placing the results into the temporary table #advance_rpt:

```
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
```

Example 8 Concatenates two columns and places the results into the temporary table #tempnames:

```
select "Author_name" = au_fname + " " + au_lname
into #tempnames
from authors
```

Example 9 Selects specific columns and rows, returns the results ordered by type from highest to lowest, and calculates summary information:

```
select type, price, advance from titles
order by type desc
compute avg(price), sum(advance) by type
compute sum(price), sum(advance)
```

Example 10 Selects specific columns and rows, and calculates totals for the price and advance columns:

```
select type, price, advance from titles
compute sum(price), sum(advance)
```

Example 11 Creates the coffeetabletitles table, a copy of the titles table which includes only books priced over \$20:

```
select * into coffeetabletitles from titles
where price > $20
```

Example 12 Creates the newtitles table, an empty copy of the titles table:

```
select * into newtitles from titles
where 1 = 0
```

Example 13 Updates the existing authors table to include only books priced over \$20:

```
select * into authors from titles
where price > $20
```

Example 14

```
select title_id, title
       from titles (index title_id_ind prefetch 16)
       where title_id like "BU%"
```

Gives an optimizer hint:

Example 15 Selects the IDENTITY column from the sales_east and sales_west tables by using the syb_identity keyword:

```
select sales_east.syb_identity,
       sales_west.syb_identity
       from sales_east, sales_west
```

Example 16 Creates the newtitles table, a copy of the titles table with an IDENTITY column:

```
select *, row_id = identity(10)
into newtitles from titles
```

Example 17 Specifies a transaction isolation level for the query.

```
select pub_id, pub_name
       from publishers
       at isolation read uncommitted
```

Example 18 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:

```
begin tran
select type, avg(price)
      from titles
      group by type
at isolation repeatable read
```

Example 19 Gives an optimizer hint for the parallel degree for the query:

```
select ord_num from salesdetail
      (index salesdetail parallel 3)
```

Example 20 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:

```
select au_id, titles.title_id, title, price
from titleauthor inner join titles
on titleauthor.title_id = titles.title_id
and price > 15
```

Example 21 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:

```
select au_fname, au_lname, pub_name
from authors left join publishers
on authors.city = publishers.city
```

Example 22 Create a new table (newtable) from the existing table (oldtable) with an identity gap, you specify it in the select into statement:

```
select identity into newtable
with identity_gap = 20
from oldtable
```

For more information about identity gaps, see the section “Managing Identity Gaps in Tables” in Chapter 7, “Creating Databases and Tables” in the *Transact-SQL User’s Guide*.

Usage

- The keywords in the select statement, as in all other statements, must be used in the order shown in the syntax statement.
- The maximum number of expressions in a select statement is 4096.
- 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.
- The character string indicated by *like* cannot be longer than 255 byte
- You cannot use the *select...for browse* option on tables containing more than 255 columns.
- 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.

Note You can select into an existing 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).
- 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)  
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 9, “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 14-2.

Table 14-2: Effects of session-level isolation levels and `readpast`

Session Isolation Level	Effects
0, read uncommitted (dirty reads)	<code>readpast</code> 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	<code>readpast</code> 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

Standard	Compliance Level	Comments
SQL92	Entry-level compliant	<p>The following are Transact-SQL extensions:</p> <ul style="list-style-type: none"> • 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 • “<i>column_heading = column_name</i>” • 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

Description Sets Adaptive Server query-processing options for the duration of the user’s work session; sets some options inside a trigger or stored procedure.

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}
```

Parameters

ansinull

determines whether evaluation of NULL-valued operands in aggregate functions is compliant with the SQL92 standard. If you use `set ansinull on`, Adaptive Server generates a warning when an aggregate function eliminates a null-valued operand from the calculation. This parameter does not affect how Adaptive Server evaluates NULL values in equality (=) or inequality (!=) comparisons.

For example, if you perform the following query on the titles table with `set ansinull off` (the default value):

```
select max(total_sales) from titles
```

Adaptive Server returns:

```
-----
      22246
```

However, if you perform the same query with `set ansinull on`, Adaptive Server returns the same value and an error message because the `total_sales` column contains NULL values:

```
-----
      22246
Warning - null value eliminated in set function
```

This message indicates that some entries in `total_sales` contain NULL instead of a real amount, so you do not have complete data on total sales for all books in this table. However, of the available data, the value returned is the highest.

ansi_permissions

determines whether SQL92 permission requirements for delete and update statements are checked. The default is off. Table 14-3 summarizes permission requirements:

Table 14-3: Permissions required for update and delete

Command	Permissions required with set ansi_permissions off	Permissions required with set ansi_permissions on
update	<ul style="list-style-type: none"> update permission on columns where values are being set 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> delete permission on table 	<ul style="list-style-type: none"> 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.

If you request character-set conversion with `set char_convert charset`, and Adaptive Server cannot perform the requested conversion, the conversion state remains the same as it was before the request. For example, if conversion is set to `off` prior to the `set char_convert charset` command, conversion remains turned off if the request fails.

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.

jtc

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 30, “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” on page 419 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_resources` is on, Adaptive Server prints the sorting plan but does not execute the create index statement. For more information, see Chapter 24, “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 34, “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_truncation

determines whether Adaptive Server raises a SQLSTATE exception when an insert or update command truncates a char, unichar, varchar or univarchar 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

Example 1 For each query, returns a description of the processing plan, but does not execute it:

```
set showplan, noexec on
go
select * from publishers
go
```

Example 2 Sets the limit on text or image data returned with a select statement to 100 bytes:

```
set textsize 100
```

Example 3 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)

```
set rowcount 4
```

Example 4 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:

```
set char_convert on with error
```

Example 5 The user executing this command now operates within the server as the login "mary" and Mary's server user ID:

```
set proxy "mary"
```

Example 6 An alternative way of stating example 5:

```
set session authorization "mary"
```

Example 7 Returns five rows for each succeeding fetch statement requested by a client using *test_cursor*:

```
set cursor rows 5 for test_cursor
```

Example 8 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:

```
set identity_insert stores_south on
go
insert stores_south (syb_identity)
values (100)
go
set identity_insert stores_south off
go
```


Example 9 Implements read-locks with each select statement in a transaction for the duration of that transaction:

```
set transaction isolation level 3
```

Example 10 Deactivates the user's System Administrator role for the current session:

```
set role "sa_role" off
```

Example 11 Tells Adaptive Server to display a warning message if you use a Transact-SQL extension:

```
set fipsflagger on
```

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.
```

Example 12 Tells Adaptive Server to evaluate NULL-valued operands of equality (=) and inequality (!=) comparisons and aggregate functions in compliance with the entry level SQL92 standard:

```
set ansinull on
```

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.

Example 13 Causes Adaptive Server to generate an exception when truncating a char, unichar or nchar string:

```
set string_rtruncation on
```

If an insert or update statement would truncate a string, Adaptive Server displays:

```
string data, right truncation
```

Example 14 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:

```
set quoted_identifier on
go
create table "!*&strange_table"
    ("emp's_name" char(10),
    age int)
go
set quoted_identifier off
go
```

Example 15 Specifies that Component Integration Services handles outbound RPC requests by default:

```
set cis_rpc_handling on
```

Example 16 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:

```
set transactional_rpc on
```

Example 17 Activates the “doctor” role. This command is used by users to specify the roles they want activated:

```
set role doctor_role on
```

Example 18 Activates the “doctor” role when the user enters the password:

```
set role doctor_role with passwd "physician" on
```

Example 19 Deactivates the “doctor” role:

```
set role doctor_role off
```

Example 20 Specifies a maximum degree of parallelism of 4 for parallel index scans and parallel table scans on nonpartitioned tables:

```
set scan_parallel_degree 4
```

Example 21 Subsequent commands in the session or stored procedure wait 5 seconds to acquire locks before generating an error message and failing:

```
set lock wait 5
```

Example 22 Subsequent commands in the session or stored procedure return an error and fail if they cannot get requested locks immediately:

```
set lock nowait
```

Example 23 Subsequent commands in the current session or stored procedure will wait indefinitely long to acquire locks:

```
set lock wait
```

Example 24 All subsequent queries in the session will run at the repeatable reads transaction isolation level:

```
set transaction isolation level 2
```

Example 25 Enables capturing abstract plans to the dev_plans group:

```
set plan dump dev_plans on
```

Example 26 Enables loading of abstract plans from the dev_plans group for queries in the current session:

```
set plan load dev_plans on
```

Example 27 Assigns this user:

- The client name alison
- The host name money1
- The application name webserver2

```
set clientname 'alison'
set clienthostname 'money1'
set clientapplname 'webserver2'
```

Usage

- 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.
 - You can update up to 1024 columns in the set clause using literals, variables, or expressions returned from a subquery.

- `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 reassign 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` on page 309. 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"
```

Using *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 14-4 lists the global variables that contain information about the session options controlled by the set command.

Table 14-4: Global variables containing session options

Global Variable	Description
<i>@@char_convert</i>	Contains 0 if character set conversion not in effect. Contains 1 if character set conversion is in effect.
<i>@@isolation</i>	Contains the current isolation level of the Transact-SQL program. <i>@@isolation</i> takes the value of the active level (0, 1 or 3).
<i>@@options</i>	Contains a hexadecimal representation of the session's set options.
<i>@parallel_degree</i>	Contains the current maximum parallel degree setting.
<i>@@rowcount</i>	Contains the number of rows affected by the last query. <i>@@rowcount</i> is set to 0 by any command that does not return rows, such as an if, update, or delete statement. With cursors, <i>@@rowcount</i> represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request. <i>@@rowcount</i> is updated even when nocount is on.
<i>@scan_parallel_degree</i>	Contains the current maximum parallel degree setting for nonclustered index scans.
<i>@@textsize</i>	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.
<i>@@tranchained</i>	Contains the current transaction mode of the Transact-SQL program. <i>@@tranchained</i> 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.
- For more information about Java in the database, see *Java in Adaptive Server Enterprise*.

Standards

Standard	Compliance Level
SQL92	Transact-SQL extension

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 14-5.

Table 14-5: 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

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

Description

Allows a Database Owner to impersonate another user.

Syntax

```
setuser ["user_name"]
```

Examples

The Database Owner temporarily adopts Mary's identity in the database in order to grant Joe permissions on authors, a table owned by Mary:

```
setuser "mary"  
go  
grant select on authors to joe  
setuser  
go
```

Usage

- 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

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

Commands: *shutdown* – *writetext*

shutdown

Description	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	<code>shutdown [srvname] [with {wait nowait}]</code>
Parameters	<p><i>srvname</i> is the logical name by which the Backup Server is known in the Adaptive Server's <code>sys.servers</code> system table. This parameter is not required when shutting down the local Adaptive Server.</p> <p>with wait is the default. This shuts down the Adaptive Server or Backup Server gracefully.</p> <p>with nowait shuts down the Adaptive Server or Backup Server immediately, without waiting for currently executing statements to finish.</p> <hr/> <p>Note Use of <code>shutdown</code> with <code>nowait</code> can lead to gaps in <code>IDENTITY</code> column values.</p> <hr/>
Examples	<p>Example 1 Shuts down the Adaptive Server from which the <code>shutdown</code> command is issued:</p> <pre>shutdown</pre> <p>Example 2 Shuts down the Adaptive Server immediately:</p> <pre>shutdown with nowait</pre> <p>Example 3 Shuts down the local Backup Server:</p> <pre>shutdown SYB_BACKUP</pre> <p>Example 4 Shuts down the remote Backup Server <code>REM_BACKUP</code>:</p> <pre>shutdown REM_BACKUP</pre>

Usage

- 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 `syssservers` table. The system procedure `sp_addserver` adds entries to `syssservers`.
 - 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

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

Description

Removes all rows from a table.

Syntax

```
truncate table [[database.]owner.]table_name
```

Parameters

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

Removes all data from the authors table:

```
truncate table authors
```

Usage

- `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

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

Description

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]
```

Parameters

- 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

Example 1 The result set includes the contents of the stor_id and stor_name columns of both the sales and sales_east tables:

```
select stor_id, stor_name from sales
union
select stor_id, stor_name from sales_east
```

Example 2 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:

```
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
```

Example 3 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:

```
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)
```

Usage

- The total number of tables that can appear on all sides of a union query is 256.
- You can use union in select statements, for example:

```
create view
select * from Jan1998Sales
union all
select * from Feb1998Sales
union all
```

- 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 15-1 lists the rules for the datatypes and the corresponding column in the result table.

Table 15-1: Resulting datatypes in union operations

Datatype of Columns in <i>union</i> 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.
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 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

Standard	Compliance Level	Comments
SQL92	Entry level compliant	<p>The following are Transact-SQL extensions:</p> <ul style="list-style-type: none"> • 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

Description

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 ][!ru|mru])]]
[, [[database.]owner.]{view_name [readpast]}
table_name [readpast]
[(index {index_name | table_name }
 [ prefetch size ][!ru|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

```

Parameters

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* | *table_name*}

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 30, “Creating and Using Abstract Plans,” in the *Performance and Tuning Guide* for more information.

Examples

Example 1 All the McBaddens in the authors table are now MacBaddens:

```
update authors
set au_lname = "MacBadden"
where au_lname = "McBadden"
```

Example 2 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:

```
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)
```

Example 3 Changes the price of the book in the `titles` table that is currently pointed to by `title_crsr` to \$24.95:

```
update titles
set price = 24.95
where current of title_crsr
```

Example 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:

```
update titles
set price = 18.95
where syb_identity = 4
```

Example 5 Updates the titles table using a declared variable:

```
declare @x money
select @x = 0
update titles
set total_sales = total_sales + 1,
   @x = price
where title_id = "BU1032"
```

Example 6 Updates rows on which another task does not hold a lock:

```
update salesdetail set discount = 40
from salesdetail readpast
where title_id like "BU1032"
and qty > 100
```

Usage

- 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`, `unichar`, `varchar`, `univarchar` 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 text 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

Standard	Compliance Level	Comments
SQL92	Entry level compliant	<p>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.</p> <p>The use of variables is a Transact-SQL extension.</p> <p>readpast is Transact-SQL extension</p>

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_helppartition, sp_helpindex, sp_unbindefault, sp_unbindrule

update all statistics

Description Updates all statistics information for a given table.

Syntax `update all statistics table_name`

Parameters *table_name*
is the name of the table for which statistics are being updated.

Examples Updates index and partition statistics for the salesdetail table:

```
update all statistics salesdetail
```

Usage	<ul style="list-style-type: none"> • 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	<table border="1"> <thead> <tr> <th>Standard</th> <th>Compliance Level</th> </tr> </thead> <tbody> <tr> <td>SQL92</td> <td>Transact-SQL extension</td> </tr> </tbody> </table>	Standard	Compliance Level	SQL92	Transact-SQL extension
Standard	Compliance Level				
SQL92	Transact-SQL extension				
Permissions	update all statistics permission defaults to the table owner and is not transferrable.				
See also	<i>Commands</i> – update statistics, update partition statistics				

update partition statistics

Description	Updates information about the number of pages in each partition for a partitioned table.
Syntax	update partition statistics <i>table_name</i> [<i>partition_number</i>]
Parameters	<p><i>table_name</i> is the name of a partitioned table.</p> <p><i>partition_number</i> 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.</p>

Usage

- 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

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

Description	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.
Syntax	<pre>update statistics <i>table_name</i> [[<i>index_name</i>] [(<i>column_list</i>)]] [using <i>step</i> values] [with consumers = <i>consumers</i>] update index statistics <i>table_name</i> [<i>index_name</i>] [using <i>step</i> values] [with consumers = <i>consumers</i>]</pre>
Parameters	<p><i>table_name</i> When used with update statistics, <i>table_name</i> is the name of the table with which the index is associated. <i>table_name</i> is required, since Transact-SQL does not require index names to be unique in a database.</p> <p><i>index_name</i> 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.</p> <p><i>column_list</i> is a comma-separated list of columns.</p> <p>using <i>step</i> 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.</p> <p>with consumers = <i>consumers</i> specifies the number of consumer processes to be used for a sort when <i>column_list</i> is provided and parallel query processing is enabled.</p> <p>index specifies that statistics for all columns in an index are to be updated.</p>
Examples	<p>Example 1 Generates statistics for the price column of the titles table:</p> <pre>update statistics titles (price) using 40 values</pre> <p>Example 2 Generates statistics for all columns in all indexes of the authors table:</p> <pre>update index statistics authors</pre> <p>Example 3 Generates statistics for all columns in the au_names_ix index of the authors table:</p>

Usage

- ```
update index statistics authors au_names_ix
```
- 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 24, “Parallel Sorting,” in the *Performance and Tuning Guide*.
  - Table 15-2 shows the types of scans performed during update statistics, the types of locks acquired, and when sorts are needed.

**Table 15-2: Locking, scans, and sorts during update statistics**

| update statistics specifying                  | Scans and sorts performed                                                                            | Locking                                                        |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| <i>Table name</i>                             |                                                                                                      |                                                                |
| 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                                           |
| <i>Table name and clustered index name</i>    |                                                                                                      |                                                                |
| 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                                           |
| <i>Table name and nonclustered index name</i> |                                                                                                      |                                                                |
| 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                                           |
| <i>Table name and column name</i>             |                                                                                                      |                                                                |
| 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

| 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

Description

Specifies the database with which you want to work.

Syntax

use *database\_name*

Parameters

*database\_name*  
is the name of the database to open.

Examples

```
use pubs2
go
```

The current database is now pubs2.

Usage

- 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

| 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    | <i>Commands</i> – create database, drop database<br><i>System procedures</i> – sp_addalias, sp_adduser, sp_modifylogin                                                                                                                                    |

## waitfor

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Syntax      | waitfor { delay <i>time</i>   time <i>time</i>   errorexit<br>  processexit   mirrorexit }                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Parameters  | <p><b>delay</b><br/>instructs Adaptive Server to wait until the specified amount of time has passed, up to a maximum of 24 hours.</p> <p><b>time</b><br/>instructs Adaptive Server to wait until the specified time.</p> <p><b><i>time</i></b><br/>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.</p> <p><b>errorexit</b><br/>instructs Adaptive Server to wait until a kernel or user process terminates abnormally.</p> <p><b>processexit</b><br/>instructs Adaptive Server to wait until a kernel or user process terminates for any reason.</p> <p><b>mirrorexit</b><br/>instructs Adaptive Server to wait for a mirror failure.</p> |
| Examples    | <p><b>Example 1</b> At 2:20 p.m., the chess table will be updated with my next move, and a procedure called <code>sendmail</code> will insert a row in a table owned by Judy, notifying her that a new move now exists in the chess table:</p> <pre>begin     waitfor time "14:20"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

```
 insert chess(next_move)
 values('Q-KR5')
 execute sendmail 'judy'
 end
```

**Example 2** After 10 seconds, Adaptive Server prints the message specified:

```
declare @var char(8)
select @var = "00:00:10"
begin
 waitfor delay @var
 print "Ten seconds have passed. Your time
 is up."
end
```

**Example 3** After any process exits abnormally, Adaptive Server prints the message specified:

```
begin
 waitfor errorexit
 print "Process exited abnormally!"
end
```

## Usage

- 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”

The following example instructs Adaptive Server to wait until 4:23 p.m:

```
waitfor time "16:23"
```

This statement instructs Adaptive Server to wait for 1 hour and 30 minutes:

```
waitfor delay "01:30"
```

- Changes in system time (such as setting the clock back for Daylight Savings Time) can delay the waitfor command.

- You can use waitfor mirrorexist within a DB-Library program to notify users when there is a mirror failure.

## Standards

| 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

## Description

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*

## Parameters

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” on page 179.

*comparison\_operator*

is one of the following:

| <b>Operator</b> | <b>Meaning</b>           |
|-----------------|--------------------------|
| =               | 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, unichar, varchar, univarchar 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” 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`, `unichar`, `univarchar`, `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 following clause would not find it because Adaptive Server converts the entry into “Jan 1, 1900 9:20AM.”:

```
where arrival_time = '9:20'
```

However, the following clause would find it:

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

**match\_string**

is a string of characters and wildcard characters enclosed in quotes. Table 15-3 lists the wildcard characters.

**Table 15-3: 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” on page 200.

**is null**

searches for null values.

**between**

is the range-start keyword. Use `and` for the range-end value. The following range is inclusive:

```
where @val between x and y
```

The following range is not:

```
x and @val < y
```

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 the following, for a values list:

```
@a = "'1', '2', '3'"
```

**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

### Example 1

```
where advance * $2 > total_sales * price
```

**Example 2** Finds all the rows in which the phone number does not begin with 415:

```
where phone not like '415%'
```

**Example 3** Finds the rows for authors named Carson, Carsen, Karsen, and Karson

```
where au_lname like "[CK]ars[eo]n"
```

**Example 4** Finds the row of the sales\_east table in which the IDENTITY column has a value of 4:

```
where sales_east.syb_identity = 4
```

**Example 5**

```
where advance < $5000 or advance is null
```

**Example 6**

```
where (type = "business" or type = "psychology")
and advance > $5500
```

**Example 7**

```
where total_sales between 4095 and 12000
```

**Example 8** Finds the rows in which the state is one of the three in the list:

```
where state in ('CA', 'IN', 'MD')
```

Usage

- 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 “Transact-SQL Functions” for information on the use of aggregate functions, and group by and having Clauses on page 280 for examples.

- Joins and subqueries are specified in the search conditions: see the *Transact-SQL User’s Guide* for full details.
- You can include up to 250 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

| 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

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | 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      | <code>while <i>logical_expression</i> [plan "<i>abstract plan</i>"]<br/>    <i>statement</i></code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Parameters  | <p><i>logical_expression</i><br/>is any expression that returns TRUE, FALSE, or NULL.</p> <p>plan "<i>abstract plan</i>"<br/>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 30, "Creating and Using Abstract Plans," in the <i>Performance and Tuning Guide</i> for more information.</p> <p><i>statement</i><br/>can be a single SQL statement, but is usually a block of SQL statements delimited by begin and end.</p> |
| Examples    | <p>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):</p> <pre>while (select avg(price) from titles) &lt; \$30 begin     select title_id, price     from titles     where price &gt; \$20     update titles     set price = price * 2 end</pre>                                     |
| Usage       | <ul style="list-style-type: none"><li>• The execution of statements in the while loop can be controlled from inside the loop with the break and continue commands.</li><li>• 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.</li></ul> <p>For example:</p>                                                                                        |

```

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

| 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

**Description** Permits minimally logged, interactive updating of an existing text or image column.

**Syntax** writetext [[*database.*]owner.]*table\_name.column\_name*  
*text\_pointer* [readpast] [with log] *data*

**Parameters** *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** **Example 1** 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:

```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
```



```

 where au_id = "409-56-7008"
writetext blurbs.copy @val with log "hello world"

```

**Example 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"

```

## Usage

- 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

| 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

# Index

## Symbols

- 114
- \* (asterisk)
  - select** and 172
- @ (at sign)
  - local variable name 187
  - procedure parameters and 260
  - rule arguments and 119
- \ (backslash)
  - character string continuation with 457
- ::= (BNF notation)
  - in SQL statements xi
- , (comma)
  - in SQL statements xi
- { } (curly braces)
  - in SQL statements xi
- = (equals sign)
  - for assigning variables 376
  - for renaming column headings 375
- ! (exclamation point)
  - error message placeholder 338
- () (parentheses)
  - in SQL statements x
- % (percent sign)
  - error message literal 340
  - error message placeholder 338
- %nn! (placeholder format) 338
- # (pound sign), temporary table identifier prefix 126
- ?? (question marks)
  - for partial characters 350
- “ ” (quotation marks)
  - literal specification of 456
- [ ] (square brackets)
  - in SQL statements xi

## Numerics

- 0 return status

- stored procedures 107
- “0x”
  - in defaults 71
  - in rules 119
- writetext** command and *image* data 460
- 2 isolation level (repeatable reads) 385

## A

- abbreviations
  - chars** for **characters**, **readtext** 349
  - exec** for **execute** 259
  - out** for **output** 101, 260
  - tran** for **transaction**, **rollback** command 367
- abstract plans
  - creating with **create plan** 98
- accent sensitivity
  - compute** and 60
  - dictionary sort order and 334
  - group by** and 292
- access
  - ANSI restrictions on tapes 256
- access, object. *See* permissions; users
- activation** keyword, **alter role** 13
- add** keyword
  - alter role** 13
  - alter table** 18, 24
- adding
  - columns to a table 16
  - constraints for tables 16
  - messages to *sysusermessages* 340
  - mirror device 205–208
  - mutually exclusive user-defined roles 13
  - objects to *tempdb* 143
  - passwords to roles 13
  - roles 117
  - rows to a table or view 295–303
  - space to a database 7–12
  - table constraints 16

- user-defined roles 117
- aggregate functions
  - group by** clause and 281, 283
  - having** clause and 282, 283
  - scalar aggregates 283
  - vector aggregates, **group by** and 284
- aggregate-free expression, grouping by 281
- aliases
  - table correlation names 378
- aliases, column
  - compute** clauses allowing 58
  - prohibited after **group by** 281, 283
- all** keyword
  - grant** 269, 277
  - group by** 281
  - negated by **having** clause 281
  - revoke** 359
  - select** 374, 388
  - union** 428
  - where** 455
- allocation map. *See* Object Allocation Map (OAM)
- allow nested triggers** configuration parameter 165
- allow\_dup\_row** option, **create index** 87
- alter database** command 7–12
  - default** keyword 7
  - dumping databases and 10
  - for load** keyword 8
  - for proxy\_update** keyword 8
  - log on** keyword 8
  - offline databases and 10
  - on** keyword 7
  - with override** keyword 8
- alter role** command 12–16
  - activation** keyword 13
  - add** keyword 13
  - drop** keyword 13
  - exclusive** keyword 13
  - membership** keyword 13
  - passwd** keyword 13
- alter table** command 16–42
  - add** keyword 18, 24
  - asc** option 25
  - check** option 23
  - clustered** constraint 20
  - constraint** keyword 20
  - default** keyword 18
  - desc** option 25
  - drop** keyword 24
  - exp\_row\_size** option 25
  - fillfactor** option 21
  - foreign key** constraint 23
  - identity** keyword 19
  - lock allpages** option 25
  - lock datapages** option 25
  - lock datarows** option 25
  - locking scheme 16
  - max\_rows\_per\_page** option 22
  - nonclustered** constraint 20
  - on** keyword 22
  - partition** clause 24
  - primary key** constraint 20
  - references** constraint 23
  - replace** keyword 24
  - reservepagegap** option 25
  - sp\_dboption** and changing lock scheme 39
  - unique** constraint 20
  - unpartition** clause 24
  - user** keyword 19
  - when is data copy required 36
- and** keyword
  - number allowed in search conditions 456
  - range-end 454
  - in search conditions 454
- ANSI tape label
  - dumpvolume** option to **dump database** 233
  - dumpvolume** option to **dump transaction** 247
  - listonly** option to **load database** 309
  - listonly** option to **load transaction** 317
- ansinull** option, **set** 398
- any** keyword
  - where** clause 455
- arguments
  - See also* logical expressions
  - numbered placeholders for, in **print** command 338, 339
  - in user-defined error messages 344
  - where** clause, number allowed 457
- arithabort** option, **set**
  - arith\_overflow** and 399
- arithignore** option, **set**
  - arith\_overflow** and 400
- as** keyword for renaming column headings 376

- asc** index option
    - alter table** command 25, 30
    - create index** command 84
    - create table** command 129
  - ascending index order, specifying 16
  - ascending indexes 25
  - ascending order, **asc** keyword 331, 383
  - asterisk (\*)
    - select** and 172
  - at** option
    - create existing table** 75
    - create proxy table** 115
    - create table** 134
    - dump database** 232
    - dump transaction** 246
    - load database** 308
    - load transaction** 316
  - at sign (@)
    - local variable name 187
    - procedure parameters and 260
    - rule arguments and 119
  - @@char\_convert** global variable 421
  - @@error** global variable
    - select into** and 391
    - stored procedures and 104
    - user-defined error messages and 340, 347
  - @@identity** global variable 301
  - @@isolation** global variable 421
  - @@langid** global variable 343
  - @@nestlevel** global variable 264
    - nested procedures and 107
    - nested triggers and 165
  - @@options** global variable 421
  - @@parallel\_degree** global variable 421
    - set parallel\_degree** and 405
  - @@rowcount** global variable 421
    - cursors and 267
    - set nocount** and 421
    - triggers and 163
  - @@scan\_parallel\_degree** global variable 421
    - set scan\_parallel\_degree** and 408
  - @@sqlstatus** global variable
    - fetch** and 267
  - @@textsize** global variable 421
    - readtext** and 350
    - set textsize** and 410
  - @@tranchained** global variable 421
  - @@version** global variable 338
  - attributes
    - remote tables 76
  - authority. *See* permissions
  - automatic operations
    - checkpoints 48
    - datatype conversion 141
    - triggers 157
- ## B
- backslash (\)
    - for character string continuation 457
  - backups
    - See also* dump, database; dump, transaction log; load, database; load, transaction log
    - disk mirroring and 206, 216
    - disk remirroring and 212
    - incremental. *See* dump, transaction log
    - master* database 10
  - Backus Naur Form (BNF) notation x, xi
  - base tables. *See* tables
  - batch processing
    - create default** and 72
    - execute** 259, 263
    - return status 355–358
    - set** options for 417
  - bcp** (bulk copy utility)
    - changing locking scheme during 41
  - begin transaction** command 43–44
    - commit** and 52
    - rollback** to 368
  - begin...end** commands 42–43
    - if...else** and 293
    - triggers and 158
  - between** keyword
    - check** constraint using 151
    - where** 454
  - binary datatypes
    - “0x” prefix 71, 119
  - binary operation, **union** 430
  - binary sort order of character sets
    - order by** and 334
  - binding

## Index

- defaults 72
  - rules 121
  - unbinding and 221
- blanks
- character datatypes and 298, 438
- blocking process 304
- blocksize** option
- dump database** 232
  - dump transaction** 247
  - load database** 308
  - load transaction** 316
- BNF notation in SQL statements x, xi
- boolean (logical) expressions
- select** statements in 294
- brackets. *See* square brackets [ ]
- branching 267
- break** command 44–45, 458–459
- browse mode
- select** 385
- B-trees, index
- fillfactor and 85
- by** row aggregate subgroup 54
- bytes
- See also* size
  - per row 27, 138
- bytes** option, **readtext** 349

## C

### canceling

- See also* **rollback** command
- command at **rowcount** 408
- duplicate updates or inserts 87
- queries with adjusted plans 406
- transactions with arithmetic errors 399
- triggers 369

### capacity option

- dump database** 233
- dump transaction** 247

### **cascade** option, **revoke** 361, 364

### cascading changes (triggers) 160

### **case** expressions 45–48, 50–51, 327–328

- null values and 47, 50, 328

### case sensitivity

- compute** and 60

### **group by** and 291

- in SQL xii
- sort order and 334

### **chained** option, **set** 400

### chained transaction mode

- commit** and 52
- delete** and 196
- fetch** and 266
- insert** and 299
- open** and 331
- update** and 437

### chains of pages

- partitions 24, 34
- unpartitioning 24

### changes, canceling. *See* **rollback** command

### changing

- See also* updating
- constraints for tables 16
- database size 7–12
- locking scheme 16, 25
- passwords for user-defined roles 16
- table constraints 16
- tables 16–42
- user-defined roles 12
- view definitions 172

### *char* datatype

- row sort order and 335

### @@*char\_convert* global variable 421

### **char\_convert** option, **set** 401

### character sets

- conversion between client and server 401
- fix\_text** upgrade after change in 180
- multibyte, changing to 180
- set char\_convert** 401

### character strings

- empty 298
- truncation 298, 410

### characters

- “0x” 119
- not converted with **char\_convert** 401

### **chars** or **characters** option, **readtext** 349

### check constraints

- column definition conflict with 152
- insert** and 298

### **check** option

- alter table** 23

- create table** 132
- checkalloc** option, **dbcc** 178
- checkcatalog** option, **dbcc** 178
- checkdb** option, **dbcc** 178
- checker, consistency. *See* **dbcc** (Database Consistency Checker)
- checkpoint** command 48–49
- checkpoint process 48–49
  - See also* recovery; savepoints
- checkstorage** option, **dbcc** 178
- checktable** option, **dbcc** 179
- checkverify** option, **dbcc** 179
- cis\_rpc\_handling** option, **set** command 401
- client
  - character set conversion 401
- clientappname** option, **set** command 402
- clienthostname** option, **set** command 402
- clientname** option, **set** command 402
- close** command 49
- close on endtran** option, **set** 402
- closing cursors 49
- clustered** constraint
  - alter table** 20
  - create table** 129
- clustered indexes
  - See also* indexes
  - creating 83
  - fillfactor** and 85
  - migration of tables to 90, 143
  - segments and 88, 90
- cntrltype** option
  - disk init** 202
  - disk reinit** 210
- coalesce** keyword, **case** 50
- collating sequence. *See* sort order
- collision of database creation requests 68
- column name
  - aliasing 344, 375
  - grouping by 281, 283
  - union** result set 431
  - views and 168
- columns
  - adding data with **insert** 297
  - adding to table 16
  - check constraints conflict with definitions of 152
  - creating indexes on 82–97
  - defaults for 71–73, 298
  - group by** and 281
  - list and **insert** 296
  - maximum number per table 27, 138
  - null values and check constraints 152
  - null values and default 73, 121
  - number allowed in **create index** command 89
  - order by** 382
  - per table 27, 138
  - permissions on 269
  - permissions revoked 360
  - rules 298
  - rules conflict with definitions of 121
  - union** of 430
  - variable-length, and sort order 335
  - views and 168
- columns per table 27, 138
- comma (,)
  - in SQL statements xi
- command execution delay. *See* **waitfor** command
- command permissions 273–275
  - See also* object permissions; permissions
  - grant all** 277
  - grant** assignment of 268–280
  - levels 273
  - revoking 360
- commands
  - create function 79
  - order-sensitive 275, 364
  - rowcount** range for 408
  - statistics io** for 409
  - statistics time** information on 409
  - Transact-SQL, summary table 1–5
- commit** command 51–53
  - begin transaction** and 43, 52
  - rollback** and 52, 368
- commit work** command. *See* **commit** command
- common keys
  - See also* foreign keys; joins; primary keys
- compact** option, **reorg** command 354
- comparing values
  - datatype conversion for 457
  - for sort order 334–335
  - in **where** clause 457
- comparison operators
  - where** clause 452

## Index

- compatibility, data
  - create default** and 72
  - of rule to column datatype 120
- compiling
  - exec with recompile** and 261
  - joins and table count 410
  - time (**statistics time**) 409
  - without execution (**noexec**) 405
- complete\_xact** option, **dbcc** 179
- Component Integration Services
  - constraints for remote servers and 20, 23
- composite indexes 83, 96
- compressed backups
  - making 232, 246
  - unloading 308, 316
- compute** clause 53–61
  - order by** and 333, 383
  - select** 383
  - without **by** 57
- conceptual (logical) tables 161, 162
- configuration parameters 4, 351
- conflicting roles 15
- connect to** command 61
- consistency check. *See* **dbcc** (Database Consistency Checker)
- constants
  - return parameters in place of 263
- constraint** keyword
  - alter table** 20
  - create table** 128
- constraints
  - adding table 16
  - changing table 16
  - create table** 144
  - cross-database 150, 227
  - dropping table 16
  - error messages 146
  - indexes created by and **max\_rows\_per\_page** 22
  - referential integrity 148
  - unique 146
- consumer process 88
- consumers** option, **update statistics** command 445
- continuation lines, character string 457
- continue** command 63–64
  - while** loop 458
- control pages for partitioned tables 34
  - updating statistics on 444
- control-of-flow language
  - begin...end** and 42
  - create procedure** and 101
- conventions
  - See also* syntax
  - Transact-SQL syntax x
  - used in the Reference Manual x
- conversion
  - columns 141
  - dates used with **like** keyword 453
  - null values and automatic 141
  - where** clause and datatype 457
- copying
  - databases with **create database** 68–70
  - the *model* database 68
  - rows with **insert...select** 296
  - tables with **select into** 390
- correlation names
  - table names 378
- corrupt indexes. *See* **reindex** option, **dbcc**
- counters, **while** loop. *See* **while** loop
- create database** command 65–71
  - default** option 65
  - disk init** and 204
  - for load** keyword 66
  - for proxy\_update** keyword 66
  - log on** keyword 66
  - on** keyword 65
  - permission 277
  - with default\_location** keyword 66
  - with override** keyword 66
- create default** command 71–74
  - batches and 72
- create existing table** command 74–79
  - datatype conversions and 77
  - defining remote procedures 78
  - mapping to remote tables 74
  - server class changes 77
- create function** command 79
- create index** command 82–97
  - index options and locking modes 95
  - insert** and 298
  - space management properties 94
- create plan** command 97
- create procedure** command 98–110



- See also* stored procedures; extended stored procedures (ESPs)
  - order of parameters in 260, 263
  - return status and 107–108
  - select \* in** 105
  - create proxy\_table** command 114–116
    - mapping proxy tables to remote tables 114
  - create role** command 116
    - grant all** and 118
  - create rule** command 118–122
  - create schema** command 122–123
  - create table** command 125–157
    - column order and 335
    - locking scheme specification 153
    - mapping proxy tables to remote tables 155
    - null values and 19, 128
    - space management properties 153
  - create trigger** command 157–175, 276, 365
  - create view** command 167–175
  - creating
    - databases 65–71
    - defaults 71–74
    - extended stored procedures 98–110
    - indexes 82–97
    - rules 118–122
    - schemas 122–123
    - tables 125–157, 376
    - tables, with identity column 153
    - triggers 157–167, 276, 365
    - user-defined roles 116
    - views 167–175
  - curly braces ({} in SQL statements xi
  - current database
    - changing 448
  - current locks, **sp\_lock** system procedure 305
  - current processes. *See* processes (server tasks)
  - cursor result set 190
    - datatypes and 266
    - returning rows 265
  - cursor rows** option, **set** 402
  - cursors
    - closing 49
    - compute** clause and 58
    - datatype compatibility 266
    - deallocating 186
    - declaring 188–192
    - deleting rows 197
    - fetching 265–267
    - grant** and 275
    - group by** and 283
    - Halloween problem 192
    - opening 330
    - order by** and 333
    - read-only 191
    - scans 190
    - select** and 390
    - union** prohibited in updatable 429
    - updatable 191
    - updating rows 439
- ## D
- damaged database, removing and repairing 180
  - data dictionary. *See* system tables
  - data integrity 298
    - See also* referential integrity constraints
    - dbcc** check for 177
  - data modification
    - text* and *image* with **writetext** 460
    - update** 432
  - database devices
    - alter database** and 7
    - new database 65
    - transaction logs on separate 207, 213
  - database dump. *See* dump, database; dump devices
  - database object owners
    - See also* database owners; ownership
  - database objects
    - adding to *tempdb* 142
    - permissions on 274
    - permissions when creating procedures 110
    - permissions when creating triggers 167
    - permissions when creating views 174
    - permissions when executing procedures 110
    - permissions when executing triggers 167
    - permissions when invoking views 174
    - referencing, **create procedure** and 105
    - select\_list* 344–345, 375–376
  - database owners
    - See also* database object owners; permissions
    - permissions granted by 269

## Index

- use of **setuser** 273
- databases
  - backing up 231–244
  - checkalloc** option (**dbcc**) 178
  - checkdb** option (**dbcc**) 178
  - checkstorage** option (**dbcc**) 178, 179
  - creating 65
  - creating with separate log segment 252
  - creation permission 71
  - default size 67
  - dropping 219
  - dumping 231–244
  - increasing size of 7
  - loading 307–315
  - number of Server 68
  - offline, altering 10
  - recovering 307–315
  - removing and repairing damaged 180
  - selecting 448
  - suspending 340
  - upgrading database dumps 312, 322
  - use** command 448
- data-only locked tables
  - restrictions for adding, dropping, or modifying columns 37
- dataserver** utility command
  - See also Utility Programs* manual
  - disk mirror** and 207
  - disk remirror** and 213
- datatype conversions
  - column definitions and 141
- datatypes
  - comparison in **union** operations 431
  - compatibility of column and default 72
  - cursor result set and 266
  - invalid in **group by** and **having** clauses 283
  - local variables and 187
- date parts
  - order of 403
- datefirst** option, **set** 402
- dateformat** option, **set** 403
- dates
  - display formats 402
  - display formats, **waitfor** command 450
- datetime* datatype
  - See also set* command
- dbcc** (Database Consistency Checker) 177–185
  - See also individual dbcc options*
  - readtext** and 350
  - dbcc traceon** 183
  - dbcc tune** 183
  - DB-Library programs
    - browse mode 385
    - dbwritetext** and **dbmoretext**, **writetext** compared to 462
    - prepare transaction** 337
    - set** options for 405, 416
    - waitfor mirrorexit** and 451
  - dbrepair** option, **dbcc** 180
  - deactivation of disk mirroring 214–217
  - deadlocks
    - descending scans and 336
  - deallocate cursor** command 186
  - deallocating cursors 186
  - debugging aids
    - set showplan on** 408
    - set sort\_resources on** 409
    - set statistics io on** 409
    - triggers and 165
  - declare** command 186–188
  - declare cursor** command 188–192
  - declaring
    - local variables 186
    - parameters 100
  - default database size** configuration parameter
    - in *sysconfigures* 67
  - default** keyword
    - alter database** 7
    - alter table** 18
    - create table** 127
  - default** option
    - create database** command 65
  - default* segment
    - extending 11
  - default settings
    - parameters for stored procedures 100
    - set** command options 415
    - weekday order 416
  - default values
    - datatypes when no length specified 100
  - defaults 298
    - column 18

- creating 71–74
- definitions and **create default** 71–73
- dropping 220
- IDENTITY columns and 35
- rules and 73, 121
- defining local variables 186–188
- degree of parallelism
  - select** and **parallel** 379
- delayed execution (**waitfor**) 449
- delete** command 193–199
  - readpast** option 193
  - triggers and 161
  - truncate table** compared to 427
- delete shared statistics** command 199
- delete statistics** command 199
- deleted* table
  - triggers and 161, 162
- deleting
  - See also* dropping
  - unlocked rows 193
- density** option
  - dump database** 232
  - dump transaction** 246
  - load database** 308
  - load transaction** 316
- dependencies, database object
  - sp\_depends** system procedure 142
- desc** index option
  - alter table** command 30
  - create index** command 84
  - create table** command 129
- desc** option
  - alter table** 25
- descending index order, specifying 16
- descending indexes 25
- descending order (**desc** keyword) 331, 383
- descending scans 335
  - deadlocks and 336
  - overflow pages and 336
- device failure
  - dumping transaction log after 249, 251
- device fragments
  - number of 68
- device initialization. *See* initializing
- devices
  - disk mirroring to 205–208
    - master 10
    - numbering 201, 209
    - secondary 207
  - dictionary sort order 334
  - dirty pages
    - updating 48–49
  - disabling mirroring. *See* disk mirroring
  - disconnect** command 61
  - disk controllers 202, 210
  - disk devices
    - adding 201–205
    - mirroring 205–208
    - unmirroring 214–217
  - disk init** command 201–205
    - master* database backup after 203
  - disk mirror** command 205–208
  - disk mirroring 205–208
    - database dump and 243
    - database load and 314
    - restarting 212–214
    - transaction log dump and 257
    - transaction log load and 324
    - unmirroring and 214–217
    - waitfor mirrorexit** 449
  - disk refit** command 208
    - create database** and 69
  - disk reinit** command 209–212
    - See also* **disk init** command
  - disk remirror** command 212–214
    - See also* disk mirroring
  - disk unmirror** command 214–217
    - See also* disk mirroring
  - dismount** option
    - dump database** 233
    - dump transaction** 248
    - load database** 309
    - load transaction** 317
- display
  - create procedure** statement text 109
  - procedures for information 102
  - setting for command-affected rows 404
- distinct** keyword
  - create view** 168
  - select** 375, 388
- distributed Transaction Processing (DTP) 180
- dividing tables into groups. *See* **group by** clause

## Index

- domain rules 298
  - create rule** command 118
  - violations 298
- “don’t recover” status of databases created **for load** 70
- doubling quotes
  - in character strings 456
- drop database** command 219–220
  - damaged databases and 180
- drop default** command 220–221
- drop index** command 222
- drop** keyword
  - alter role** 13
  - alter table** 24
- drop procedure** command 223–224
  - grouped procedures and 223, 260
- drop role** command 224
- drop rule** command 225–226
- drop table** command 226–228
- drop trigger** command 228–229
- drop view** command 229–230
- dropdb** option, **dbcc dbrepair** 180
- dropping
  - constraints for tables 16
  - corrupt indexes 182
  - damaged database 180
  - databases 219–220
  - dbcc dbrepair** database 180
  - defaults 72, 220
  - grouped procedures 99
  - indexes 222
  - passwords from roles 13
  - procedures 223–224
  - roles in a mutually exclusive relationship 13
  - rows from a table 193–199, 227
  - rows from a table using **truncate table** 427
  - rules 225
  - table constraints 16
  - tables 226–228
  - tables with triggers 162
  - triggers 162, 228
  - user-defined roles 225
  - views 229
- dump database**
  - compress** option 232
- dump database** command 231–244
  - See also* dump, database
    - after using **create database** 69
    - after using **disk init** 203
    - after using **dump transaction with no\_log** 246
    - dump transaction** and 237
    - master* database and 237
    - select into** and 391
- dump devices
  - See also* database devices; log device
  - dump, database and 232
  - dump, transaction log and 246
  - naming 232, 246, 253–254
  - number required 313
- dump striping
  - database dumps and 233
  - transaction dumps and 247
- dump transaction**
  - compress** option 246
- dump transaction** command 244–258
  - See also* dump, transaction log
  - after using **disk init** 203
  - permissions for execution 258
  - select into/bulkcopy/pilsort** and 250
  - standby\_access** option 249
  - trunc log on chkpt** and 250
  - with no\_log** option 252–253
  - with no\_truncate** option 249, 251
  - with truncate\_only** option 251
- dump, database
  - across networks 237
  - appending to volume 242–243
  - Backup Server and 239
  - Backup Server, remote 232
  - block size 232
  - commands used for 251
  - dismounting tapes 233
  - dump devices 232, 238
  - dump striping 233
  - dynamic 237
  - expiration date 234
  - file name 234, 239
  - initializing/appending 234
  - loading 69, 307–315
  - master* database 238
  - message destination 235
  - new databases and 237
  - overwriting 234, 242–243

- remote 239
- rewinding tapes after 234
- scheduling 237–238
- successive 242, 256
- system databases 238
- tape capacity 233
- tape density 232
- thresholds and 238
- volume changes 242
- volume name 233, 241
- dump, transaction log
  - across networks 253
  - appending dumps 248
  - appending to volume 256–257
  - Backup Server, remote 254
  - command used for 251
  - dismounting tapes 248
  - dump striping 247
  - expiration date 248
  - file name 248, 254–255
  - initializing tape 248
  - initializing volume 256–257
  - insufficient log space option 252–253
  - loading 315–324
  - message destination 249
  - permissions problems 250
  - remote 254, 256
  - rewinding tapes after 248
  - scheduling 253
  - tape capacity 247
  - thresholds and 253
  - volume name 247, 255
- dumpvolume** option
  - dump database** 233
  - dump transaction** 247
  - load database** 308
  - load transaction** 316
- duplicate rows
  - indexes and 83, 87
  - removing with **union** 428
- duplication
  - of space for a new database 69
  - of a table with no data 391
- dynamic dumps 237, 253
- dynamic execution of Transact-SQL commands 259

## E

- else** keyword. *See if...else* conditions
- empty string (“ ”) or ( ‘ ’ )
  - as a single space 298
  - updating an 437
- enable xact coordination** configuration parameter 411
- end** keyword 42
- engine** option, **dbcc** 180
- @@error** global variable
  - select into** and 391
  - stored procedures and 104
  - user-defined error messages and 340, 347
- error handling
  - in character set conversion 401
  - dbcc** and 185
  - triggers and 165
- error messages
  - 12207 325, 326
  - character conversion 401
  - printing user-defined 340
  - user-defined 342–347
- errorexit** keyword, **waitfor** 449
- errors
  - See also* error messages; SQLSTATE codes
  - allocation 178, 181, 183
  - datatype conversion 127
  - numbers for user-defined 343
  - return status values 357
- escape** keyword
  - where** 453
- evaluation order 430
- exception report, **dbcc tablealloc** 181, 183
- exclamation point (!)
  - error message placeholder 338
- exclusive** keyword
  - alter role** 13
- exclusive** option, **lock table** 325
- execute** command 259–265
  - create procedure** and 104
- executing
  - extended stored procedures 259
  - procedures 259
  - Transact-SQL commands 259
  - user-defined procedures 259
- execution

## Index

- specifying times for 449
  - execution delay. *See* **waitfor** command
  - exists** keyword
    - where** 454
  - exit
    - unconditional, and **return** command 355–358
    - waitfor** command 449
  - exp\_row\_size** option
    - create table** 133, 154
    - select into** 377
    - setting before **alter table...lock** 33
    - specifying with **create table** 133
    - specifying with **select into** 377
  - explicit values for IDENTITY columns 300, 404
  - expressions
    - evaluation order in 430
    - grouping by 283
    - insert** and 296
    - summary values for 58
  - extended columns, Transact-SQL 285, 287
  - extended stored procedures
    - C run-time signals not allowed 105
    - creating 98–110
    - dropping 223
    - executing 259
  - extending
    - database storage 7
  - Extensions, Transact-SQL 285
  - extents 90
    - create table** and 138
    - dbcc indexalloc** report on index 181
    - dbcc** report on table 182
  - external** option
    - create existing table** 74
    - create proxy\_table** 114
    - create table** 134
- ## F
- failures, media
    - See also* recovery
    - automatic failover and 215
    - disk remirror** and 213
  - fast** option
    - dbcc indexalloc** 181
    - dbcc tablealloc** 181, 183
  - fetch** command 265–267
  - fetching cursors 265–267
  - file names
    - database dumps 239
    - DLL 101
    - listing database dump with **listonly** 309
    - listing transaction log with **listonly** 317
    - transaction log dumps 248, 317
  - file** option
    - dump database** 234
    - dump transaction** 248
    - load database** 309
    - load transaction** 317
  - files
    - See also* tables; transaction log
    - mirror device 206
  - fillfactor
    - create index** and 85
  - fillfactor** option
    - alter table** 21
    - create index** 85, 94
    - create table** 130, 153
  - fillfactor** values
    - alter table...lock** 31
  - FIPS flagger
    - insert** extension not detected by 303
    - set** option for 403
    - update** extensions not detected by 442
  - fipsflagger** option, **set** 403
  - first column parameter. *See* keys
  - fix** option
    - dbcc** 178, 181, 183
    - dbcc indexalloc** 181
    - dbcc tablealloc** 178
  - fix\_text** option, **dbcc** 180, 185
  - fixed-length columns
    - stored order of 335
  - flushmessage** option, **set** 403
  - for browse** option, **select** 385
    - union** prohibited in 431
  - for load** keyword
    - alter database** 8
    - create database** command 66
  - for load** option
    - create database** 69

**for proxy\_update** keyword  
     **alter database** 8  
     **create database** command 66  
**for read only** option, **declare cursor** 188  
**for update** option, **declare cursor** 188  
**forceplan** option, **set** 403  
 forcing offline pages online 237  
**foreign key** constraint  
     **alter table** 23  
     **create table** 132  
 foreign keys 146  
**forget\_xact** option, **dbcc** 180  
 format strings  
     **print** 338  
     **raiserror** 343  
     in user-defined error messages 343  
**forwarded\_rows** option, **reorg** command 354  
 fragmentation, reducing 16  
**from** keyword  
     **delete** 193  
     **grant** 273  
     **load database** 308  
     **load transaction** 316  
     **select** 378  
     **update** 433  
**full** option  
     **dbcc indexalloc** 181  
     **dbcc tablealloc** 181, 182

## G

german language print message example 338  
**goto** keyword 267–268  
 grammatical structure, numbered placeholders and 338  
 grand totals  
     **compute** 57  
     **order by** 334  
**grant** command 62, 268–280  
     **all** keyword 269  
     **drop role** permission not included in 225  
     public group and 270  
     roles and 278  
**grant option for** option, **revoke** 361  
 granting

    create trigger permission 166, 276, 365  
**group by** clause 280–292  
     aggregate functions and 281, 283  
     **having** clause and 280–292  
     **having** clause and, in standard SQL 284  
     **having** clause and, in Transact-SQL 285  
     **having** clause and, sort orders 291  
     **select** 381–382  
     views and 173  
     without **having** clause 291  
 grouping  
     multiple trigger actions 158  
     procedures of the same name 99, 223, 260  
     table rows 284  
 groups  
     See also “public” group  
     **grant** and 279  
     **revoke** and 365  
     table rows 280  
 guest users  
     permissions 279

## H

halloween problem 192  
**having** clause 280–292  
     aggregate functions and 282, 283  
     **group by** and 280–292  
     **group by** extensions in Transact-SQL and 285  
     negates **all** 281  
     **select** 382  
 headings, column 281  
     in views 168  
 heuristic completion 179  
 hexadecimal numbers  
     “0x” prefix for 71  
 hierarchy of permissions. *See* permissions  
 histograms  
     specifying steps with **create index** 94  
     specifying steps with **update statistics** 445  
**holdlock** keyword  
     **readtext** 348  
     **select** 380

- I**
- I/O
- devices, disk mirroring to 206
  - displaying total actual cost (**statistics io**) 409
  - prefetch and **delete** 194
  - prefetch and **select** 379
  - prefetch and **update** 433
- identifiers
- select** 389
- identities
- sa\_role** and Database Owner 418
  - set proxy** and 419
  - set session authorization** and 419
  - setuser** command 423
- identity burning set factor** configuration parameter 300
- IDENTITY columns
- adding, dropping, or modifying with **alter table** 38
  - creating tables with 152
  - defaults and 35
  - inserting values into 296
  - inserts into tables with 300
  - maximum value of 300
  - null values and 301
  - selecting 301, 392
  - updates not allowed 439
  - views and 172
- identity gap
- setting 153
- @*identity* global variable 301
- identity** keyword
- alter table** 19
  - create table** 127
- identity of user. *See* aliases; logins; users
- identity\_insert** option, **set** 404
- IDs, user
- stored procedure (**procid**) 406
- if update** clause, **create trigger** 158, 164
- if...else** conditions 293–295
- continue** and 63
  - local variables and 187
- ignore\_dup\_key** option, **create index** 87
- ignore\_dup\_row** option, **create index** 87
- image* datatype
- length of data returned 389, 410
  - order by** not allowed 333
  - pointer values in **readtext** 347
  - storage on separate device 348
  - triggers and 161
  - union** not allowed on 431
  - writetext** to 460
- immediate shutdown 425
- impersonating a user. *See* **setuser** command
- in** keyword
- alter table** and 24
  - check** constraint using 151
  - where** 454
- inactive transaction log space 246
- included groups, **group by** query 285
- incremental backups. *See* dump, transaction log
- index keys
- asc** option for ordering 91
  - desc** option for ordering 91
  - maximum number of bytes 89
  - number of 89
  - ordering 91
- index pages
- fillfactor effect on 21, 85, 130
  - leaf level 21, 83, 85, 130
- indexalloc** option, **dbcc** 181
- indexes
- ascending 25
  - composite 96
  - creating 82–97
  - dbcc indexalloc** and 181
  - descending 25
  - dropping 222
  - integrity checks (**dbcc**) 182
  - joins and 89
  - key values 446
  - listing 222
  - max\_rows\_per\_page** and 22, 131
  - naming 83
  - nonclustered 83
  - number allowed 89
  - Object Allocation Maps of 181
  - page allocation check 181
  - specifying order of 16
  - specifying sort order with **alter table** 30
  - specifying sort order with **create index** 91
  - specifying sort order with **create table** 144
  - truncate table** and 427
  - types of 83



- update index statistics** on 445
- update statistics** on 89, 445
- views and 90
- infected processes
  - waitfor errorexit** and 450
- information (server)
  - display procedures 102
  - space usage 96
  - text 109
- information messages (server). *See* error messages; severity levels, error
- init** option
  - dump database** 234
  - dump transaction** 248
- initializing
  - disk reinit** and 203, 209–212
  - disk space 201–205
- in-memory map 10
- insert** command 295–303
  - create default** and 72
  - IDENTITY columns and 300
  - null/not null columns and 172
  - triggers and 161, 164
  - update** and 297
  - views and 172, 301–302
- inserted* table
  - triggers and 161, 162
- integrity of data
  - constraints 144
  - methods 145
- integrity. *See* **dbcc** (Database Consistency Checker); referential integrity
- internal datatypes of null columns 141
- interval, automatic checkpoint 48
- into** keyword
  - fetch** 265
  - insert** 295
  - select** 376, 390
  - union** 429
- is null** keyword
  - where** 453
- isnull** system function
  - insert** and 299
  - print** and 340
  - select** and 389
- @*isolation* global variable 421

- isolation levels
  - readpast** option and 395
  - repeatable reads 385

## J

- Japanese character sets
  - print message example 338
- Java columns, adding 37
- Java items
  - remove java** command 352
- joins
  - indexes and 89
  - number of tables considered by optimizer 410
  - table groups and 287
- jtc** option, **set** 404

## K

- key columns
  - dropping with **alter table** 37
- key values 446
- keys, index. *See* index keys
- keys, table 146
  - See also* common keys; indexes
- kill** command 303–305

## L

- labels
  - dump volumes 241, 314, 323
  - goto** label 267
  - @*langid* global variable 343
  - language** option, **set** 404
- languages, alternate
  - structure and translation 338
  - system messages and 404
  - weekday order and 416
- leaf levels of indexes
  - clustered index 21, 83, 85, 130
- leaving a procedure. *See* **return** command
- levels
  - nested procedures and 107, 264

## Index

- nesting triggers 165
  - @@nestlevel 107
- permission assignment 273
- like** keyword
  - alter table** and 24
  - check** constraint using 151
  - where** 453
- listing
  - existing defaults 220
  - user group members 279
- listonly** option
  - load database** 309
  - load transaction** 317
- lists
  - commands 1–5
  - error return values 358
  - reserved return status value 358
  - sort order choices and effects 334
- load database**
  - compress** option 308
- load database** command 307–315
- load transaction**
  - compress** option 316
- load transaction** command 315–324
- load, database 307–315
  - across networks 313
  - Backup Server and 314
  - block size 308
  - cross-platform not supported 311, 319
  - disk mirroring and 314
  - dismounting tapes after 309
  - file name, listing 309
  - header, listing 310
  - load striping 309
  - message destination 310, 323
  - new database 69
  - remote 313
  - restricting use 312, 322
  - rewinding tapes after 309
  - size required 312
  - updates prohibited during 312
  - volume name 308
- load, transaction log 315–324
  - disk mirroring and 324
  - dismounting tape after 317
  - dump devices 316
  - file name, listing 317
  - header, listing 318
  - load striping 316
  - message destination 318
  - point-in-time recovery 318
  - rewinding tape after 317
  - until\_time** 318
  - volume name 316
- local variables
  - declare** (name and datatype) 186
  - raiserror** and 344
  - in screen messages 338
  - in user-defined error messages 344
- location of new database 65
- lock allpages** option
  - alter table** 25
  - create table** command 133
  - select into** command 376
- lock datapages** option
  - alter table** 25
  - create table** command 133
  - select into** command 376
- lock datarows** option
  - alter table** 25
  - alter table** command 40
  - create table** command 133
  - select into** command 376
- lock nowait** option, **set lock** command 405
- lock table** command 324, 325
- lock wait** option, **set** command 405
- locking
  - tables with **lock table** command 324
  - text for reads 348
- locking scheme
  - changing 16, 25
  - changing with **alter table** 16
  - create table** and 153
  - modifying 25
  - specifying with **select into** 376
- locks
  - deletes skipping locked rows 193
  - selects skipping locked rows 393
  - updates skipping locked rows 432
- log device
  - See also* transaction logs
  - purging a 238

- space allocation 69, 185
  - log on** keyword
    - alter database** 8
    - create database** 66
  - log segment
    - dbcc checktable** report on 179
    - not on its own device 179
  - logging
    - select into** 390
    - text* or *image* data 460
    - triggers and unlogged operations 162
    - writetext** command 460
  - logical (conceptual) tables 161, 162
  - logical consistency. *See* **dbcc** (Database Consistency Checker)
  - logical device name
    - disk mirroring 205
    - disk remirroring 212
    - disk unmirroring 214
    - new database 65
  - logical expressions
    - if...else** 293
    - syntax 44
    - when...then** 45, 50, 327
  - logical reads (**statistics io**) 409
  - logins
    - See also* remote logins; users
    - char\_convert** setting for 401
    - disabling 426
  - logs. *See* segments; transaction logs
  - loops
    - break** and 44
    - continue** and 64
    - goto** label 268
    - trigger chain infinite 165
    - while** 44, 458
  - lowercase letters, sort order and 334
- M**
- making compressed backups 232, 246
  - mapping
    - system and default segments 11
  - markers, user-defined. *See* placeholders; savepoints
  - master* database
    - See also* recovery of *master* database; databases
    - alter database** and 10
    - backing up 251
    - create database** and 69
    - disk init** and 203
    - disk mirror** and 206
    - disk refit** and 208
    - disk reinit** and 209
    - disk remirror** and 212
    - disk unmirror** and 216
    - dropping databases and 219
    - transaction log purging 238, 251
  - master device 10
  - max\_rows\_per\_page** option
    - alter table** 22, 31
    - create index** 86, 94
    - create table** 131, 153
    - select into** 376
  - maximum number of columns 27, 138
  - maximum row size 27, 138
  - membership** keyword
    - alter role** 13
  - memory
    - See also* space
    - releasing with **deallocate cursor** 186
  - messages
    - language setting for 404
    - printing user-defined 337–340
    - revoke** 364
    - screen 337–340
    - trigger 161, 229
  - migration
    - of system log to another device 204
    - of tables to clustered indexes 90, 143
  - mirror** keyword, **disk mirror** 206
  - mirrorexit** keyword
    - waitfor** 449
  - mistakes, user. *See* errors
  - mode** option, **disk unmirror** 215
  - model* database
    - copying the 68
  - modifying
    - databases 7
    - locking scheme 25
    - roles 12
    - tables 16

- multibyte character sets
    - changing to 180
    - fix\_text** upgrade for 180, 185
    - readtext** and 350
    - readtext using characters** for 350
    - writetext** and 462
  - multicolumn index. *See* composite indexes
  - multiple trigger actions 158
  - multitable views 441
    - See also* views
    - delete** and 172, 195
  - mutually exclusive roles 13
- ## N
- name of device
    - disk mirroring and 205
    - disk remirroring and 212
    - disk unmirroring and 214
    - dump device 232, 246
    - physical, **disk reinit** and 209
    - remote dump device 313
  - name** option
    - disk init** 201
    - disk reinit** 209
  - names
    - alias for table 378
    - column, in views 168
    - parameter, in **create procedure** 100
    - segment 22, 88, 131, 134
    - setuser** 423
    - sorting groups of 291
    - view 229
  - naming
    - columns in views 168
    - cursors 189
    - database device 201
    - file 201
    - indexes 83
    - stored procedures 105
    - tables 126
    - temporary tables 142
    - triggers 157
    - views 168
  - nested **select** statements. *See* **select** command; subqueries
  - nesting
    - begin...end** blocks 42
    - if...else** conditions 295
    - levels 107
    - levels of triggers 165
    - stored procedures 105, 264
    - triggers 165
    - while** loops 459
    - while** loops, **break** and 44
  - @*nestlevel* global variable 264
    - nested procedures and 107
    - nested triggers and 165
  - %*nn!* (placeholder format) 338
  - no\_log** option, **dump transaction** 246
  - no\_truncate** option, **dump transaction** 249
  - nocount** option, **set** 404
  - nodismount** option
    - dump database** 233
    - dump transaction** 248
    - load database** 309
    - load transaction** 317
  - noexec** option, **set** 405
  - nofix** option, **dbcc**
    - checkalloc** and 178
    - indexalloc** and 181
    - tablealloc** and 183
  - noholdlock** keyword, **select** 348, 380
  - noinit** option
    - dump database** 234
    - dump transaction** 248
  - nonclustered** constraint
    - alter table** 20
    - create table** 129
  - nonclustered indexes 83
  - noserial** option, **disk mirror** 206
  - not** keyword
    - where** 451
  - not null** keyword
    - create table** 19, 128
  - not null values
    - dropping defaults for 221
    - insert** and 299
    - select** statements and 389
    - views and 172

- notify** option
    - dump database** 235
    - dump transaction** 249
    - load database** 310
    - load transaction** 318
  - nounload** option
    - dump database** 234
    - dump transaction** 248
    - load database** 309
    - load transaction** 317
  - nowait** option
    - lock table** command 325
    - set lock** command 405
  - nowait** option, **shutdown** 425
  - null** keyword
    - create table** 19, 127, 128
  - null values
    - check constraints and 152
    - column defaults and 73, 121
    - defining 73, 141
    - dropping defaults for 221
    - group by** and 283
    - inserting substitute values for 299
    - new column 73
    - new rules and column definition 121
    - null defaults and 73, 121
    - select** statements and 389
    - sort order of 334
    - stored procedures cannot return 358
    - text* and *image* columns 298
    - triggers and 164
  - nullif** expressions 327–328
  - nullif** keyword 327
  - number (quantity of)
    - active dumps or loads 239, 254, 313, 323
    - and** and **or** conditions allowed 456
    - arguments and placeholders 339
    - arguments, in a **where** clause 457
    - bytes in returned text 350
    - bytes per row 27, 138
    - clustered indexes 83
    - columns for index key 89
    - databases Server can manage 68
    - device fragments 68
    - different triggers 161
    - having** clause search arguments 282
    - logical reads (**statistics io**) 409
    - named segments 68
    - nesting levels 107
    - nesting levels, for triggers 165
    - nonclustered indexes 83, 89
    - parameters in a procedure 187
    - physical reads (**statistics io**) 409
    - placeholders in a format string 339
    - scans (**statistics io**) 409
    - steps for distribution histogram 87
    - stored procedure parameters 105
    - tables allowed in a query 378
    - tables per database 138
    - updates 165
    - user-defined roles 118
  - number of columns
    - in an **order by** clause 333
    - per table 27, 138
    - in a view 171
  - number of pages
    - in an extent 90, 138
    - statistics io** and 409
    - written (**statistics io**) 409
  - numbers
    - error return values (Server) 357
    - placeholder (%*nn!*) 338
    - procid** setting 406
    - same name group procedure 99, 223, 260
    - select list 382
    - statistics io** 409
    - virtual device 201, 204, 209
    - weekday names and 402
- ## O
- Object Allocation Map (OAM) pages
    - dbcc indexalloc** and 181
    - dbcc** report on table 182
  - object names, database
    - as parameters 100
    - in stored procedures 107, 109
  - object owners. *See* database object owners
  - object permissions
    - See also* command permissions; permissions
    - grant** 268–280

## Index

- grant all** 277
  - of option, **declare cursor** 188
  - offline databases and **alter database** command 10
  - offset position, **readtext** command 348
  - offsets** option, **set** 405
  - on** keyword
    - alter database** 7
    - alter table** 22
    - create database** command 65
    - create index** 88, 90
    - create table** 131, 134
  - online database** command 313, 328, 328–330
    - bringing databases online 313
    - dump transaction** and 320
    - load transaction** and 319
    - upgrades and 322
  - Open Client applications
    - keywords 405
    - procid** setting 406
    - set** options for 405, 416
  - open** command 330–331
  - opening cursors 330
  - optdiag** utility
    - loading simulated statistics 200, 421
    - overwriting statistics with **create index** 94
  - optimized** report
    - dbcc indexalloc** 181
    - dbcc tablealloc** 182
  - optimizer
    - join selectivity 410
  - @@options** global variable 421
  - or** keyword
    - number allowed in search conditions 456
    - where** 455
  - order
    - of arguments in translated strings 338
    - ascending sort 331, 383
    - of column list and insert data 296
    - of columns (fixed- and variable-length) 335
    - of creating indexes 90
    - of date parts 403
    - descending sort 331, 383
    - error message arguments 338
    - of evaluation 430
    - of names in a group 291
    - of null values 334
    - of parameters in **create procedure** 260, 263
    - for unbinding a rule 120
  - order by** clause 331–337
    - compute by** and 58, 333, 383
    - select** and 382
  - order of commands 275, 364
  - original identity, resuming an (**setuser** command) 423
  - output
    - dbcc** 185
    - zero-length string 340
  - output** option
    - create procedure** 101, 260, 263
    - execute** 260
    - return parameter 260
  - overflow errors
    - set arithabort** and 399
  - overhead
    - triggers 162
  - override. *See* **with override** option
  - overwriting triggers 161, 229
  - owners. *See* database object owners; database owners
  - ownership
    - See also* permissions; **setuser** command
    - of command and object permissions 273
    - of rules 121
    - of stored procedures 110
    - of triggers 167
    - of views 175
- ## P
- padding, data
    - blanks and 298
  - page splits 22, 86, 131
  - pages
    - ratio of filled to empty 16
  - pages, control
    - updating statistics on 444
  - pages, data
    - See also* index pages; table pages
    - chain of 24, 34–35
    - extents and 91, 138
    - extents and **dbcc tablealloc** 182
    - extents reported by **dbcc indexalloc** 181
    - multibyte characters and 180

- statistics io** and 409
- pages, index
  - truncate table** and 427
- pages, OAM (Object Allocation Map)
  - dbcc indexalloc** report on 181
  - dbcc** report on table 182
- pages, overflow
  - descending scans and 336
- pair, mirrored 214
- parallel** keyword, **select** command 379
- @*parallel\_degree* global variable 421
  - set parallel\_degree** and 405
- parallel\_degree** option, **set** command 405
- parameters, procedure
  - datatypes 100
  - defaults 100
  - execute** and 260
  - naming 100
  - not part of transactions 264
  - ways to supply 260, 263
- parentheses ()
  - in SQL statements x
- parseonly** option, **set** 405
- partial characters, reading 350
- partition** clause, **alter table** command 24
- partition statistics
  - updating with **update partition statistics** 444
  - updating with **update statistics** 443
- partitioned tables
  - alter table** 24
- partitioning
  - tables 16
- passthrough mode
  - connect to** command 62
- passwd** keyword
  - alter role** 13
- passwords
  - adding to roles 13
  - adding to user-defined roles 15
  - changing for user-defined roles 16
  - dropping from roles 13
  - dropping from user-defined roles 15
  - roles and 12
  - user-defined roles and 116, 407
- path name
  - DLL and extended stored procedures 101
  - mirror device 206
  - remote dump device 313
- percent sign (%)
  - error message placeholder 338
  - literal in error messages 340
- performance
  - select into** and 391
  - showplan** and diagnostics 408
  - sort\_resources** and diagnostics 409
  - triggers and 162
  - writetext** during **dump database** 462
- permissions
  - assigned by database owner 269
  - assigning 269
  - changing with **setuser** 423
  - command 273–275
  - creating with **create schema** 122–123
  - for creating triggers 166, 276, 365
  - grant** 268–280
  - object 274
  - “public” group 273–275
  - revoke** command 359–366
- physical database consistency. *See* **dbcc** (Database Consistency Checker)
- physical reads (**statistics io**) 409
- physname** option
  - disk init** 201
  - disk reinit** 209
- placeholders
  - print** message 338
- plan
  - create procedure** and 101
  - set showplan on** and 408
  - set sort\_resources on** and 409
- plans
  - creating with **create plan** 98
- pointers
  - text* or *image* column 348
- pointers, device. *See* segments
- pound sign (#) temporary table name prefix 126
- precedence
  - order-sensitive commands and 275, 364
  - rule binding 121
  - of user-defined return values 358
- preference, uppercase letter sort order 334
- prefetch** keyword

## Index

- delete** 194
  - select** 379
  - set** 406
  - update** 433
  - prepare transaction** command 337
  - primary key** constraint
    - alter table** 20
    - create table** 129
  - primary keys 146
  - updating 159
  - primary** option, **disk unmirror** 214
  - print** command 337–340
    - local variables and 188
    - using **raiserror** or 340
  - printing user-defined messages 337–340
  - privileges. *See* permissions
  - procedure groups 223, 260
  - procedure** option
    - create existing table** 74
  - procedure plan, **create procedure** and 101
  - procedures. *See* stored procedures; system procedures
  - process logical name. *See* logical device name
  - process\_limit\_action** option, **set** 406
  - processes (server tasks)
    - See also* servers
    - ID number 304
    - infected, **waitfor errorexit** 450
    - killing 303–305
    - sp\_who** report on 304
  - processexit** keyword, **waitfor** 449
  - procid** option, **set** 406
  - protection system
    - command and object permissions 273
    - hierarchy of roles, groups and users 279
    - stored procedures 110
    - user-defined roles 117
  - proxy** option, **set** 407
    - granting 270
    - revoking 360
  - proxy tables
    - mapping to remote tables 74
    - mapping to remote tables with **create proxy\_table** 114
    - mapping to remote tables with **create table** 155
  - “public” group 279, 365
    - See also* groups
  - grant** and 270
    - permissions 273–275
  - revoke** and 361
  - public** keyword
    - grant** 270
    - revoke** 361
- ## Q
- queries
    - compilation without execution 405
    - execution settings 396–423
    - keywords list 405
    - syntax check (**set parseonly**) 405
    - trigger firing by 160
    - union** 428–432
    - views and 171
    - with/without **group by** and **having** 284
  - query analysis
    - set noexec** 405
    - set statistics io** 409
    - set statistics time** 409
  - query plans
    - set showplan on** and 408
  - query processing
    - set** options for 396
  - question marks (??)
    - for partial characters 350
  - quiesce database** command 340–342
  - quotation marks (“ ”)
    - literal specification of 456
  - quoted\_identifier** option, **set** 407
- ## R
- raiserror** command 342–347
    - compared to **print** 347
    - local variables and 188
    - using **print** or 340
  - range
    - set rowcount** 408
  - ratio of filled to empty pages 16
  - read-only cursors 191
  - readpast** option



- delete** command 194
- isolation levels and 395
- readtext** command 348
- select** command 377
- update** command 433
- writetext** command 460
- readtext** command 347–351
- rebuild** option, **reorg** command 354
- rebuild\_text** option, **dbcc** 181
- rebuilding
  - automatic, of nonclustered index 90
  - indexes 182
  - system tables 181, 183
  - text and image data 181
- reclaim\_space** option, **reorg** command 354
- recompilation
  - create procedure with recompile** option 101, 105
  - execute with recompile** option 261
  - stored procedures 105
- reconfigure** command 351
- recovery
  - dump transaction** and 253
  - to specified time in transaction log 320
  - time and **checkpoint** 48
- recovery of *master* database 238
  - after using **create database** 69
  - after using **disk init** 203
- re-creating
  - indexes 182
  - procedures 108
  - tables 226
  - text and image data 181
- recursions, limited 165
- reducing
  - storage fragmentation 16
- reference information
  - Transact-SQL commands 1–5
- references** constraint
  - alter table** 23
  - create table** 132
- referencing, object. *See* dependencies, database object
- referential integrity
  - triggers for 157–167
- referential integrity constraints 16, 148, 237
  - create table** and 144
  - cross-database 150, 227
- regulations
  - sort order ties 334–335
- reindex** option, **dbcc** 182
- reinitializing, **disk reinit** and 209–212
- remirroring. *See* disk mirroring
- remote procedure calls 389
  - execute** and 264
  - rollback** and 368
- remote procedures, defining 78
- remote servers 389
  - constraints for 20, 23
- remove java** command 352–353
- remove** option, **disk unmirror** 215
- removing. *See* dropping; deleting
- renaming
  - identity of object owner 273
  - stored procedures 105
  - triggers 162
  - views 171
- reorg** command 353–355
- repairing a damaged database 180
- repeatable reads isolation level 385
- repeated execution. *See* **while** loop
- replace** keyword, **alter table** 24
- reports
  - sp\_who** 304
  - types of **dbcc** 182
- reserved return status values 357
- reservepagegap** option
  - alter table** 25, 31
  - create index** 86, 94
  - create table** 133, 154
  - select into** 377
- restarting **while** loops 64
- restarts, Server
  - after using **disk refit** 208
  - before using **create database** 67
  - using **dataserver** utility 207, 213
- restoring
  - See also* recovery
  - a damaged *master* database 208, 209
  - database with **load database** 307–315
- results
  - See also* output
  - of aggregate operations 284

## Index

- cursor result set 190, 265
    - order by** and sorting 331–337
  - resume** option, **reorg** 354
  - retain** option, **disk unmirror** 215
  - retaindays** option
    - dump database** 234
    - dump transaction** 248
  - retrieving
    - error message text 337
  - return** command 355–358
  - return parameters
    - output** keyword 101, 260
  - return status
    - stored procedure 259, 355
  - revoke** command 359–366
    - object and command permissions 273
    - public group and 361
  - revoking
    - create trigger permission 166, 276, 365
    - role privileges using **with override** 224
  - role** option
    - grant** 271
    - revoke** 361
    - set** command 407
  - roles
    - adding passwords to 13
    - creating (user-defined) 116
    - dropping passwords from 13
    - granting 278
    - mutually exclusive 13
    - permissions and 279
    - stored procedure permissions and 278
    - turning on and off with **set role** 407
  - roles, system
    - revoking 361
  - roles, user-defined
    - limitations 118
    - revoking 361
    - turning on and off 407
  - rollback** command 367–369
    - begin transaction** and 43
    - commit** and 52
    - triggers and 163, 165
  - rollback transaction** command. *See* **rollback** command
  - rollback trigger** command 163, 369–370
  - rollback work** command. *See* **rollback** command
  - rolling back processes
    - checkpoint** and 48
    - parameter values and 264
  - row aggregates
    - compute** and 53
  - row length 27, 138
  - row size 27, 138
  - @*rowcount* global variable 421
    - cursors and 267
    - set nocount** and 421
    - triggers and 163
  - rowcount** option, **set** 408
  - rows, table
    - See also* **select** command
    - aggregate functions applied to 284
    - comparison order of 335
    - create index** and duplication of 83, 87
    - deleting unlocked 193
    - deleting with **truncate table** 427
    - displaying command-affected 404
    - grouping 281
    - insert** 297
    - rowcount** setting 408
    - scalar aggregates applied to 284
    - selecting unlocked 393
    - update** 432
    - updating unlocked 432
    - ways to group 284
  - rules
    - binding 121
    - column definition conflict with 121
    - creating new 118–122
    - default violation of 73
    - dropping user-defined 225
    - insert** and 298
    - naming user-created 119
  - running a procedure with **execute** 259
- ## S
- save transaction** command 370–372
  - savepoints
    - See also* checkpoint process
    - rollback** and 367
    - setting using **save transaction** 371

- scalar aggregates
  - group by** and 283
- `@@scan_parallel_degree` global variable 421
  - set scan\_parallel\_degree** and 408
- scan\_parallel\_degree** option, **set** 408
- scans
  - cursor 190
  - number of (**statistics io**) 409
- schemas 122–123
  - permissions 123
- search conditions
  - group by** and **having** query 282, 285
  - select** 381
  - where** clause 451–457
- secondary** option, **disk unmirror** 214
- security
  - See also* permissions
  - command and object permissions 273
  - views and 170
- seed values
  - set identity\_insert** and 404
- segments
  - See also* database devices; log segment; space allocation
  - changing table locking schemes 40
  - clustered indexes on 90
  - creating indexes on 22, 88, 90, 131
  - dbcc checktable** report on 179
  - dbcc indexalloc** report on 181
  - mapping to a new device 11
  - names of 22, 131, 134
  - number of named 68
  - placing objects on 88
  - separation of table and index 90, 143
- select** command 373–396
  - altered rows and 27, 35
  - create procedure** and 105
  - create view** and 168
  - group by** and **having** clauses 280
  - insert** and 299
  - local variables and 187
  - size of *text* data to be returned with 410
  - triggers and 161
  - variables and 186
- select into** command 376–391
  - not allowed with **compute** 58, 383
  - select into/bulkcopy/pilsort** database option
    - select into** and 391
    - transaction log dumping and 250
- select list 344–345, 375–376
  - order by** and 382
  - union** statements 430
- select** option, **create view** 168
- selecting
  - unlocked rows 393
- self\_recursion** option, **set** 166, 408
- sentence order and numbered placeholders 338
- separation, physical
  - of table and index segments 90, 143
  - of transaction log device 207, 213
- sequence. *See* **order by** clause; sort order
- serial** option, **disk mirror** 206
- server process ID number. *See* processes (server tasks)
- servers
  - See also* processes (server tasks); remote servers
  - capacity for databases 68
- session authorization** option, **set** 408
  - revoking 270, 360
- set** command 396–423
  - See also* individual **set** options
  - default settings 415
  - inside a stored procedure 109
  - inside a trigger 162
  - lock wait** 405
  - roles and 407
  - statistics simulate** 409
  - strict\_dtm\_enforcement** 409
  - transaction isolation level** 411
  - within **update** 433
- setting
  - identity gap 153
- setuser** command 423–424
  - user impersonation using 273
- severity levels, error
  - user-defined messages 346
- share** option, **lock table** 325
- shared** keyword
  - select** 380
- showplan** option, **set** 408
- shutdown** command 425–427
- side** option, **disk unmirror** 214
- size

## Index

- columns in table 27, 138
- compiled stored procedure 105
- composite index 84
- database extension 8
- estimation of a compiled stored procedure 105
- image* data to be returned with **writetext** 461
- initialized database device 204
- log device 204
- new database 66
- readtext** data 348, 350
- recompiled stored procedures 105
- row 27, 138
- set textsize** function 410
- tables 138
- text* data to be returned with **select** 410
- text* data to be returned with **writetext** 461
- transaction log device 69, 204
- size limit
  - columns allowed per table 138
  - order by** results 333
  - print** command 339
  - tables per database 138
- size option
  - disk init** 202, 210
- skip\_ncindex** option, **dbcc** 178
- sort operations (**order by**)
  - sorting plan for 409
- sort order
  - See also* order
  - ascending 331
  - choices and effects 333
  - descending 331
  - group by** and **having** and 291
  - groups of names 291
  - order by** and 334
  - rebuilding indexes after changing 182
  - specifying index with **alter table** 30
  - specifying index with **create index** 91
  - specifying index with **create table** 144
- sort\_merge** option, **set** 408
- sort\_resources** option, **set** 409
- sp\_bindefault** system procedure
  - create default** and 72
- sp\_bindrule** system procedure
  - create rule** and 120
- sp\_dboption** system procedure
  - checkpoints and 48
- sp\_depends** system procedure 142
- sp\_transactions** system procedure 180
- sp\_unbindefault** system procedure 220
- sp\_unbindrule** system procedure
  - create rule** and 120
  - drop rule** and 225
- space
  - See also* size; space allocation
  - adding to database 7–12
  - for a clustered index 21, 85, 91, 130
  - clustered indexes and **max\_rows\_per\_page** 22, 86
  - database storage 21, 85, 91, 130
  - dbcc checktable** reporting free 179
  - extents 90, 138
  - extents for indexes 181
  - freeing with **truncate table** 427
  - for index pages 21, 85, 130
  - max\_rows\_per\_page** and 22, 86, 131
  - new database 66
  - for recompiled stored procedures 105
  - required for **alter table...lock** 40
  - required for **reorg rebuild** 355
  - retrieving inactive log 246
  - running out of 246
  - for stored procedures 105
  - used on the log segment 179, 246
- space allocation
  - dbcc** commands for checking 178–181
  - log device 69
  - pages 182
  - table 138, 178
- space management properties
  - create index** and 94
  - create table** and 153
- space reclamation
  - reorg reclaim\_space** for 353
- spaces, character
  - update** of 438
- speed (Server)
  - create database for load** 69
  - create index with sorted\_data** 87
  - dump transaction** compared to **dump database** 253
  - execute** 263
  - truncate table** compared to **delete** 427

- writetext** compared to **dbwritetext** and **dbmoretext** 462
- SQL standards
  - set** options for 422
  - set session authorization** and 408
- @*sqlstatus* global variable
  - fetch** and 267
- square brackets [ ]
  - in SQL statements xi
- standby\_access** option
  - dump transaction** 249
  - online database** 328
- starting Servers
  - disk mirroring of master device and 207
  - disk remirroring of master device and 213
- startserver** utility command
  - See also Utility Programs* manual
  - disk mirror** and 207
  - disk remirror** and 213
- statements
  - create trigger** 158
  - in **create procedure** 101
- statistics
  - deleting table and column with **delete statistics** 200
  - generating for unindexed columns 446
  - simulated, loading 200, 421
- statistics** clause, **create index** command 87
- statistics io** option, **set** 409
- statistics simulate** option, **set** command 409
- statistics subquerycache** option, **set** 409
- statistics time** option, **set** 409
- status
  - stored procedures execution 264
- stopping
  - procedures. *See return* command
  - servers 425
- storage fragmentation, reducing 16
- stored procedure triggers. *See* triggers
- stored procedures
  - creating 98–110
  - dropping 99, 223–224
  - dropping groups 223
  - executing 259
  - grouping 99, 260
  - ID numbers 406
  - naming 99
  - nesting 105, 264
  - parseonly** not used with 405
  - permissions granted 269
  - permissions revoked 360
  - procid** option 406
  - renaming 105
  - return status 107–108, 259, 264, 355
  - set** commands in 396
  - storage maximums 105
- strict dtm enforcement** configuration parameter 409
- strict\_dtm\_enforcement** option, **set** command 409
- string\_rtruncation** option, **set** 410
  - insert** and 298
  - update** and 438
- strings
  - print** message 338
  - truncating 298, 438
- stripe on** option
  - dump database** 233
  - dump transaction** 247
  - load database** 309
  - load transaction** 316
- structure
  - See also order*
  - clustered** and **nonclustered** index 83
- subgroups, summary values for 57
- subqueries
  - order by** and 333
  - union** prohibited in 431
- summary values
  - generation with **compute** 57
- suspect indexes
  - See also reindex* option, **dbcc**
- suspending databases 340
- syb\_identity** keyword
  - select** and 392
- sybsecurity* database
  - dropping 220
- symbols
  - in SQL statements x
- synonyms
  - chars** for **characters**, **readtext** 349
  - out** for **output** 101, 260
  - tran**, **transaction**, and **work**, **commit** command 51
  - tran**, **transaction**, and **work**, **rollback** command

## Index

- 367
  - syntax
    - check using **set parseonly** 405
  - syntax conventions, Transact-SQL x
  - syscolumns* table 178
  - syscomments* table
    - default definitions in 72
    - procedure definitions in 109
    - rule definitions in 121
    - trigger definitions in 166, 174
  - sysconfigures* table
    - database size** parameter 67
  - sysdevices* table
    - disk init** and 203
    - mirror names in 215
  - sysindexes* table
    - composite indexes and 96
  - syslogs* table
    - See also* recovery; transaction logs
    - put on a separate device 207, 213
    - running **dbcc checktable** on 179
  - sysmessages* table
    - raiserror** and 343
  - sysobjects* table
    - trigger IDs and 166
  - sysprocedures* table
    - trigger execution plans in 166
  - sysprotects* table
    - grant/revoke** statements and 276, 364
    - sp\_changegroup** and 279
  - sys.servers* table
    - Backup Server and 239, 254
    - load database** and 313
  - sysstatistics* table
    - removing statistics with **delete statistics** 199
  - system activities
    - setting query-processing options for 396–423
    - shutdown** 425
  - system databases
    - dumping 238
  - system logical name. *See* logical device name
  - system messages
    - See also* error messages; messages
    - language setting for 404
  - system messages, language setting for 404
  - system procedures
    - See also* **create procedure** command; *individual procedure names*
    - create procedure** and 98–110
    - dropping user-defined 223–224
  - system roles
    - revoking 361
    - stored procedures and 278
  - system* segment
    - alter database** 11
  - system tables
    - See also* tables; *individual table names*
    - affected by **drop table** 226
    - affected by **drop view** 229
    - dbcc checkcatalog** and 178
    - default definitions in 72
    - fixing allocation errors found in 181, 183
    - lock table** prohibited on 326
    - rebuilding of 181, 183
    - rule information in 119
    - triggers and 161
  - sys.transactions* table 179
  - sysusermessages* table
    - raiserror** and 343
- ## T
- table count** option, **set** 410
  - table** option
    - create table** 134
  - table pages
    - allocation with **dbcc tablealloc** 182
  - tablealloc** option, **dbcc** 182
  - tables
    - allowed in a **from** clause 378
    - changing 16–42
    - creating duplicate 391
    - creating new 125–157, 376
    - creating with **create schema** 122–123
    - creating with identity column 153
    - dbcc checkdb** and 178
    - dividing, with **group by** and **having** clauses 280–292
    - dropping 226–228
    - external 114
    - index location 222, 446

- migration to a clustered index 90, 143
- with no data 391
- number considered in joins 410
- Object Allocation Maps of 182
- partitioning 16, 24, 34–35
- permissions on 269
- permissions revoked 360
- proxy 74
- single-group 284
- Transact-SQL extension effects and querying 285
- unpartitioning 16, 24
- tape labels
  - listonly** option to **load database** 309
  - listonly** option to **load transaction** 317
- tempdb* database
  - adding objects to 143
  - sysobjects* table and 142
  - systypes* table and 143
- temporary tables
  - create procedure** and 109
  - create table** and 126, 142
  - identifier prefix (#) 126
  - indexing 89
  - lock table** prohibited on 326
  - naming 142
- text* datatype
  - initializing with **update** 438
  - length of data returned 389, 410
  - order by** not allowed 333
  - storage on separate device 348
  - textsize** setting 410
  - triggers and 161
  - union** not allowed on 431
- text pointer values
  - readtext** and 348
- textptr** function 348, 350
- @@*textsize* global variable 421
  - readtext** and 350
  - set textsize** and 410
- textsize** option, **set** 410
- then** keyword. *See* **when...then** conditions
- thresholds
  - database dumps and 238
  - transaction log dumps and 253
- ties, regulations for sort order 334–335
- time interval
  - See also* timing
  - automatic checkpoint 48
  - elapsed execution (**statistics time**) 409
  - reorg** 354
  - for running a trigger 162
  - waitfor** 449
- time** option
  - reorg** 354
  - waitfor** 449
- timestamps, order of transaction log dumps 313
- timing
  - See also* time interval
  - automatic checkpoint 48
- to** option
  - dump database** 232
  - dump transaction** 246
  - revoke** 364
- totals
  - compute** command 334
  - @@*tranchained* global variable 421
  - transaction isolation level** option, **set** 410
- transaction isolation levels
  - readpast** option and 395
- transaction logs
  - See also* **dump transaction** command; *syslogs* table
  - backing up 231
  - of deleted rows 196
  - dump database** and 231
  - dumping 245
  - inactive space 246
  - loading 315–324
  - master* database 238, 251
  - placing on separate segment 252
  - purging 238
  - on a separate device 204, 207, 213, 250
  - space extension 11
  - space, monitoring 253
  - syslogs* table **trunc log on chkpt** 250
  - writetext with log** and 460
- transactional\_rpc** option, **set** 411
- transactions
  - See also* batch processing; **rollback** command; user-defined transactions
  - begin** 43
  - canceling. *See* **rollback** command

- chained 52
  - dump transaction** command 244–258
  - ending with **commit** 51
  - fetch** and 266
  - isolation levels 411
  - parameters not part of 264
  - preparing 337
  - save transaction** and 370–372
  - update** iteration within given 437
  - Transact-SQL commands
    - executing 259
    - extensions for 285
    - summary table 1–5
  - translation
    - of arguments 338
  - trigger tables 163
  - triggers
    - creating 157–167, 276, 365
    - delete** and 197
    - dropping 228
    - enabling self-recursion 166
    - insert** and 298
    - nested 165–166
    - nested, and **rollback trigger** 369
    - @*nestlevel* and 165
    - on *image* columns 161
    - on *text* columns 161
    - parseonly** not used with 405
    - recursion 166
    - renaming 162
    - rollback** in 163, 368
    - rolling back 369
    - @*rowcount* and 163
    - self-recursion 166
    - set** commands in 396
    - stored procedures and 165
    - system tables and 161
    - time interval 162
    - truncate table** command and 427
    - update** and 435
  - truncate table** command 427–428
    - delete** triggers and 162
    - faster than **delete** command 196
  - truncate\_only** option, **dump transaction** 246, 251
  - truncation
    - datatypes with no length specified 100
    - default values 72
    - insert** and 298
    - log, prohibited on mixed device 66
    - set string\_rtruncation** and 410
    - spaces to a single space 438
    - transaction log 244
- ## U
- unbinding
    - defaults 72, 220
    - rules 225
  - unconditional branching to a user-defined label 267
  - undoing changes. *See* **rollback** command
  - union** operator 428–432
    - maximum number of tables 429
    - restrictions on use 431
  - unique constraints 146
  - unique** keyword
    - alter table** 20
    - create index** 83
    - create table** 128
  - unload** option
    - dump database** 234
    - dump transaction** 248
    - load database** 309
    - load transaction** 317
  - unloading compressed backups 308, 316
  - unmirroring devices. *See* disk mirroring
  - unpartition** clause, **alter table** 24
  - unpartitioning
    - tables 16
  - updatable cursors 191
  - update all statistics** command 442, 445
  - update** command 432–442
    - ignore\_dup\_key** and 87
    - ignore\_dup\_row** and 92
    - insert** and 297
    - readpast** option 433
    - triggers and 161
    - triggers and **if update** 164
    - views and 172, 441
  - update index statistics** command 445
  - update partition statistics** command 443–444
  - update statistics** command 445–448



- create index** and 89
  - locking during 446
  - scan type 446
  - sort requirements 446
- updating
  - data in views 172
  - “dirty” pages 48–49
  - ignore\_dup\_key** and 87
  - primary keys 159
  - trigger firing by 166
  - unlocked rows 432
  - writetext** 460
- uppercase letter preference 334
- us\_english language
  - weekdays setting 416
- use** command 448–449
- user errors. *See* errors; severity levels
- user groups. *See* groups; “public” group
- user** keyword
  - alter table** 19
  - create table** 127
- user permissions. *See* database owners; permissions
- user-defined procedures
  - creating 98–110
  - executing 259
- user-defined roles
  - adding passwords to 13
  - conflicting 15
  - creating 116
  - revoking 361
  - system procedures and 278
  - turning on and off 407
- user-defined transactions
  - See also* transactions
  - begin transaction** 43
  - ending with **commit** 51
- users
  - guest permissions 279
  - impersonating (**setuser**) 273
  - system procedure permissions and 275
- using** option, **readtext** 349, 350
- using...values** option, **update statistics** command 445

## V

- values
  - IDENTITY columns 300
  - procedure parameter or argument 260
- values** option, **insert** 296
- varchar** datatype
  - spaces in and **insert** 298
- variable-length columns
  - empty strings in 298
  - stored order of 335
- variables
  - assigning as part of a select list 376
  - in **update** statements 435
  - local 186–188
  - in **print** messages 338
  - return values and 263
- vdevno** option
  - disk init** 201
  - disk reinit** 209
- vector aggregates
  - group by** and 284
- @@*version* global variable 338
- views
  - See also* database objects; multitable views
  - allowed in a **from** clause 378
  - changes to underlying tables of 172
  - check option** and 440–441
  - creating 167–175
  - creating with **create schema** 122–123
  - dropping 229
  - inserting data through 301
  - permissions on 269, 274
  - permissions revoked 360
  - readtext** and 350
  - renaming 171
  - update** and 172, 440–441
  - updating restrictions 441
  - with check option** 172, 301–302
  - violation of domain or integrity rules 298
  - virtual device number 201, 204, 209
  - volume names, database dumps 241

## W

- wait** option, **lock table** command 325

## Index

- wait** option, **shutdown** 425
- waitfor** command 449–451
- waiting for **shutdown** 426
- weekday date value
  - names and numbers 402
- when** keyword. *See* **when...then** conditions
- when...then** conditions 45
- where** clause 451–457
  - aggregate functions not permitted in 456
  - delete** 193
  - group by** clause and 285
  - having** and 456
  - repeating a 288
- where current of** clause
  - delete** 194
  - update** 434
- while** keyword 458–459
  - continue** and 63
  - exiting loop with **break** 44
  - loops 458
- with check option** option
  - create view** 169
  - views and 174
- with consumers** clause, **create index** 88
- with consumers** option, **update statistics** command 445
- with default\_location** keyword
  - create database** command 66
- with grant option** option, **grant** 271
- with** keyword
  - rollback trigger** 369
  - set role** command 407
- with log** option, **writetext** 460
- with no\_error** option, **set char\_convert** 401
- with no\_log** option, **dump transaction** 246
- with no\_truncate** option, **dump transaction** 249
- with nowait** option, **shutdown** 425
- with override** keyword
  - alter database** 8
  - create database** command 66
- with override** option 224
- with recompile** option
  - create procedure** 101
  - execute** 261
- with resume** option, **reorg** 354
- with standby\_access** option
  - dump transaction** 249
  - with statistics** clause, **create index** command 87
  - with time** option, **reorg** 354
  - with truncate\_only** option, **dump transaction** 246, 251
  - with wait** option, **shutdown** 425
- work session, **set** options for 396–423
- write operations
  - logging *text or image* 460
- writes** option, **disk mirror** 206
- writetext** command 460–462
  - triggers and 162

## X

X/Open XA 180

## Z

zero-length string output 340