SYBASE®

Sybase® Adaptive Server™ Enterprise
Reference Manual
Volume 3: Datatypes and System Tables

Adaptive
Server™

Document ID: 32416-01-1150

September 1997

Copyright Information

Table of Contents

Chapters 1 through 6 are in Volumes 1 and 2 of the *Adaptive Server Reference Manual.*

About This Book

7. System and User-Defined Datatypes

8. System Tables

A. Expressions, Identifiers, and Wildcard Characters

B. Reserved Words

C. SQLSTATE Codes and Messages

Index

List of Tables

About This Book

The *Adaptive Server Reference Manual* is a three-volume guide to Sybase® Adaptive Server™ Enterprise and the Transact-SQL® language. This volume includes information about datatypes, system tables, reserved words, SQLSTATE errors, and an index for all three volumes. Volume 1, *Commands and Functions*, contains information about Transact-SQL commands and built-in functions. Volume 2, *Procedures*, includes information about system procedures, catalog stored procedures, system extended stored procedures, and **dbcc** stored procedures.

For information about the intended audience of this book, related documents, other sources of information, conventions used in this manual, and help, refer to "About This Book" in Volume 1.

How to Use This Book

This manual consists of the following:

- Chapter 7, "System and User-Defined Datatypes," describes the system and user-defined datatypes that are supplied with Adaptive Server and indicates how to use them to create user-defined datatypes.
- Chapter 8, "System Tables," contains information about all system tables in the *master* database, the auditing database, and any user databases (such as *pubs2*).
- Appendix A, "Expressions, Identifiers, and Wildcard Characters" contains information about using the Transact-SQL language.
- Appendix B, "Reserved Words," contains information about the Transact-SQL and SQL92 keywords.
- Appendix C, "SQLSTATE Codes and Messages," contains information about Adaptive Server's SQLSTATE status codes and the associated messages.
- The Index contains entries for all three volumes of the *Adaptive Server Reference Manual*.

System and User-Defined Datatypes

7 System and User-Defined Datatypes

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

Datatype Categories

Adaptive Server provides a number of system datatypes, as well as the user-defined datatypes *timestamp* and *sysname*.

Adaptive Server datatypes fall into the categories listed in Table 7-1. Each category is described in a section of this chapter.

Table 7-1: Datatype categories

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

Range and Storage Size

Table 7-2 lists the system-supplied datatypes and their synonyms and provides information about the range of valid values and storage size for each. For simplicity, the datatypes are printed in lowercase characters, although Adaptive Server allows you to use either uppercase or lowercase characters for system datatypes. (User-defined datatypes, such as *timestamp*, are **case sensitive**.) Most Adaptive Server-supplied datatypes are not reserved words and can be used to name other objects.

Table 7-2: Range and storage size for system datatypes

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

Table 7-2: Range and storage size for system datatypes (continued)

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

Declaring the Datatype of a Column, Variable, or Parameter

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

Declaring the Datatype for a Column in a Table

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

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

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

For example:

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

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

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

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

For example:

```
   declare @hope money
```

Declaring the Datatype for a Parameter in a Stored Procedure

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

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

For example:

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

Determining the Datatype of a Literal

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

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

➤ ***Note***

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

Datatype of Mixed-Mode Expressions

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

Determining the Datatype Hierarchy

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

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

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

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

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

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

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

```
smallint(qty) * int(royalty) = int
```

Determining Precision and Scale

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

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

Adaptive Server determines the precision and scale of the results as shown in Table 7-3:

Table 7-3: Precision and scale after arithmetic operations

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

Converting One Datatype to Another

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

Automatic Conversion of Fixed-Length NULL Columns

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

Table 7-4 lists the fixed- and variable-length datatypes to which they are converted. Certain variable-length datatypes, such as *moneyn*, are reserved datatypes; you cannot use them to create columns, variables, or parameters:

Table 7-4: Automatic conversion of fixed-length datatypes

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

Handling Overflow and Truncation Errors

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

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

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

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

➤ ***Note***

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

Standards and Compliance

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

Exact Numeric Datatypes

Function

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

Integer Types

Adaptive Server provides three exact numeric datatypes, *tinyint*, *smallint*, and *int* (or *integer*), to store integers. Choose among the integer types based on the expected size of the numbers to be stored. Internal storage size varies by type, as shown in Table 7-5:

Table 7-5: Integer datatypes

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

Entering Integer Data

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

Table 7-6 shows some valid entries for a column with a datatype of *integer* and indicates how **isql** displays these values:

Table 7-6: Valid integer values

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

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

Table 7-7: Invalid integer values

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

Decimal Datatypes

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

Specifying Precision and Scale

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

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

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

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

Storage Size

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

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

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

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

Entering Decimal Data

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

Table 7-8 shows some valid entries for a column with a datatype of *numeric*(5,3) and indicates how these values are displayed by **isql**:

Table 7-8: Valid decimal values

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

Table 7-9 shows some invalid entries for a column with a datatype of *numeric*(5,3):

Table 7-9: Invalid decimal values

Value Entered	Type of Error
1,200	Commas not allowed.
12-	Minus sign should precede digits.
12.345678	Too many nonzero digits to the right of the decimal point.

Standards and Compliance

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

Approximate Numeric Datatypes

Function

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

Understanding Approximate Numeric Datatypes

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

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

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

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

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

Range, Precision, and Storage Size

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

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

Table 7-10 shows the range and storage size for each approximate numeric type. Note that **isql** displays only 6 significant digits after the decimal point and rounds the remainder:

Table 7-10: Approximate numeric datatypes

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

Entering Approximate Numeric Data

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

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

The value represented by the entry is the following product:

$\texttt{mantissa * 10}^{\text{EXPONENT}}$

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

Standards and Compliance

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

Money Datatypes

Function

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

Accuracy

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

Range and Storage Size

Table 7-11 summarizes the range and storage requirements for money datatypes:

Table 7-11: Money datatypes

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

Entering Monetary Values

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

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

Standards and Compliance

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

timestamp Datatype

Function

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

Datatype Definition

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

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

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

Creating a *timestamp* Column

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

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

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

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

or to a column with another name:

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

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

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

Standards and Compliance

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

Date / time Datatypes

Function

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

➤ ***Note***

Adaptive Server also provides a *timestamp* datatype, which stores binary-type information.

Range and Storage Requirements

Table 7-12 summarizes the range and storage requirements for the *datetime* and *smalldatetime* datatypes:

Table 7-12: Transact-SQL datatypes for storing dates and times

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

Entering *datetime* and *smalldatetime* Data

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

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

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

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

- You can enter the entire date as an unseparated string of 4, 6, or 8 digits, or use slash(/), hyphen (-), or period(.) separators between the date parts.
  - When entering dates as unseparated strings, use the appropriate format for that string length. Use leading zeros for single-digit years, months, and days. Dates entered in the wrong format may be misinterpreted or result in errors.
  - When entering dates with separators, use the **set dateformat** option to determine the expected order of date parts. If the first date part in a separated string is four digits, Adaptive Server interprets the string as *yyyy-mm-dd* format.
- Some date formats accept 2-digit years (*yy*). Dates greater than or equal to 50 are interpreted as 19yy; those less than 50 are interpreted as 20yy.
- You can specify the month as either a number or a name. Month names and their abbreviations are language-specific and can be entered in uppercase, lowercase, or mixed case.
- If you omit the date portion of a *datetime* or *smalldatetime* value, Adaptive Server uses the default date of January 1, 1900.

Table 7-13 describes the acceptable formats for entering the date portion of a *datetime* or *smalldatetime* value:

Table 7-13: Date formats for datetime and smalldatetime datatypes

Date Format	Interpretation	Sample Entries	Meaning
4-digit string with no separators	Interpreted as *yyyy*. Date defaults to Jan 1 of the specified year.	“1947”	Jan 1 1947
6-digit string with no separators	Interpreted as *yymmdd*. For *yy* < 50, year is 20*yy*. For *yy* >= 50, year is 19*yy*.	“450128” “520128”	Jan 28 2045 Jan 28 1952
8-digit string with no separators	Interpreted as *yyyymmdd*.	“19940415”	Apr 15 1994

Table 7-13: Date formats for datetime and smalldatetime datatypes (continued)

Date Format	Interpretation	Sample Entries	Meaning
String consisting of 2-digit month, day, and year separated by slashes, hyphens, or periods, or a combination of the above.	The **dateformat** and **language set** options determine the expected order of date parts. For us_english, the default order is *mdy*. For *yy* < 50, year is interpreted as 20*yy*. For *yy* >= 50, year is interpreted as 19*yy*.	"4/15/94" "4.15.94" "4-15-94" "04.15/94"	All of these entries are interpreted as Apr 15 1994 when the **dateformat** option is set to **mdy**.
String consisting of 2-digit month, 2-digit day, and 4-digit year separated by slashes, hyphens, or periods, or a combination of the above.	The **dateformat** and **language set** options determine the expected order of date parts. For us_english, the default order is *mdy*.	"04/15.1994"	Interpreted as Apr 15 1994 when the **dateformat** option is set to **mdy**.
Month is entered in character form (either full month name or its standard abbreviation), followed by an optional comma.	If 4-digit year is entered, date parts can be entered in any order.	"April 15, 1994" "1994 15 apr" "1994 April 15" "15 APR 1994"	All of these entries are interpreted as Apr 15 1994.
	If day is omitted, all 4 digits of year must be specified. Day defaults to the first day of the month.	"apr 1994"	Apr 1 1994
	If year is only 2 digits (*yy*), it is expected to appear after the day. For *yy* < 50, year is interpreted as 20*yy*. For *yy* >= 50, year is interpreted as 19*yy*.	"mar 16 17" "apr 15 94"	Mar 16 2017 Apr 15 1994
The empty string, ""	Date defaults to Jan 1 1900.	""	Jan 1 1900

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

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

```
hours[:minutes[:seconds[:milliseconds]] [AM | PM]
```

- Use 12AM for midnight and 12PM for noon.
- A time value must contain either a colon or an AM or PM signifier. The AM or PM can be entered in uppercase, lowercase, or mixed case.

- The seconds specification can include either a decimal portion preceded by a decimal point or a number of milliseconds preceded by a colon. For example, “12:30:20:1” means twenty seconds and one millisecond past 12:30; “12:30:20.1” means twenty and one-tenth of a second past 12:30.
- If you omit the time portion of a *datetime* or *smalldatetime* value, Adaptive Server uses the default time of 12:00:00:000AM.

Display Formats for *datetime* and *smalldatetime* Values

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

Table 7-14 lists some examples of *datetime* entries and their display values:

Table 7-14: Examples of datetime entries

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

Finding *datetime* Values That Match a Pattern

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

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

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

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

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

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

When searching with **like**, you cannot use the wide variety of input formats that are available for entering the date portion of *datetime* and *smalldatetime* values. Since the standard display formats do not include seconds or milliseconds, you cannot search for seconds or milliseconds with **like** and a match pattern, unless you are also using *style* 9 or 109 and the **convert** function.

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

```
like May 2%
```

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

Manipulating Dates

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

Standards and Compliance

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

Character Datatypes

Function

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

Length and Storage Size

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

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

Fixed-length columns tend to take more storage space than variable-length columns, but are accessed somewhat faster. Table 7-15 summarizes the storage requirements of the different character datatypes:

Table 7-15: Character datatypes

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

Table 7-15: Character datatypes (continued)

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

Determining Column Length with System Functions

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

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

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

Entering Character Data

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

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

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

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

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

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

Treatment of Blanks

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

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

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

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

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

(5 rows affected)
```

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

- Only *char* **not null** and *nchar* **not null** columns are padded to the full width of the column; *char* **null** columns are treated like *varchar* and *nchar* **null** columns are treated like *nvarchar*.
- Preceding blanks are not affected.
- Trailing blanks are truncated except for *char* and *nchar* **not null** columns.

- The empty string ("") is treated as a single space. In *char* and *nchar* **not null** columns, the result is a column-length field of spaces.

Manipulating Character Data

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

Standards and Compliance

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

Binary Datatypes

Function

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

Valid Binary and Varbinary Entries

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

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

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

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

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

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

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

Treatment of Trailing Zeros

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

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

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

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

select * from zeros

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

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

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

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

select * from zeros where bnot = 0x00

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

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

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

insert sample values ('002710000000ae1b')

select * from sample

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

Platform Dependence

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

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

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

Standards and Compliance

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

bit Datatype

Function

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

Entering Data into *bit* Columns

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

Storage Size

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

Restrictions

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

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

sysname Datatype

Function

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

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

Using the *sysname* Datatype

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

Standards and Compliance

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

text and image Datatypes

Function

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

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

Defining a *text* or *image* Column

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

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

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

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

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

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

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

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

Putting Additional Pages on Another Device

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

Zero Padding

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

➤ ***Note***

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

Partitioning Has No Effect on How the Data Is Stored

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

Initializing *text* and *image* Columns

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

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

```
create table texttest
(title_id varchar(6), blurb text null, pub_id
char(4))
insert texttest values
("BU7832", "Straight Talk About Computers is an
annotated analysis of what computers can do for
you: a no-hype guide for the critical user.",
"1389")
```

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

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

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

➤ ***Note***

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

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

Saving Space by Allowing Nulls

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

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

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

Getting Information from *sysindexes*

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

Table 7-16: Storage of text and image data

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

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

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

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

Using *readtext* and *writetext*

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

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

Determining How Much Space a Column Uses

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

```
     sp_spaceused blurbs

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

Restrictions on *text* and *image* Columns

text and *image* columns cannot be used:

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

Selecting *text* and *image* Data

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

Table 7-17: text and image global variables

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

Converting the *text* and *image* Datatypes

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

Pattern Matching in *text* Data

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

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

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

Duplicate Rows Are Prohibited

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

Standards and Compliance

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

User-Defined Datatypes

Function

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

Creating Frequently Used Datatypes in the *model* Database

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

Creating a User-Defined Datatype

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

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

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

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

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

Renaming a User-Defined Datatype

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

Dropping a User-Defined Datatype

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

➤ ***Note***

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

Getting Help on Datatypes

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

Standards and Compliance

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

System Tables

8 System Tables

This chapter describes the Adaptive Server system tables.

All tables in the *master* database are system tables. Some of these tables also occur in user databases—they are automatically created when the **create database** command is issued.

Lists of System Tables

The following system tables occur in all databases:

System Table	Contents
sysalternates	One row for each Adaptive Server user mapped to a database user
sysattributes	One row for each object attribute definition.
syscolumns	One row for each column in a table or view, and for each parameter in a procedure
syscomments	One or more rows for each view, rule, default, trigger, and procedure, giving SQL definition statement
sysconstraints	One row for each referential and check constraint associated with a table or column
sysdepends	One row for each procedure, view, or table that is referenced by a procedure, view, or trigger
sysgams	Allocation bitmaps for an entire database
sysindexes	One row for each clustered or nonclustered index, and one row for each table with no indexes, and an additional row for each table containing text or image data
syskeys	One row for each primary, foreign, or common key; set by user (not maintained by Adaptive Server)
syslogs	Transaction log
sysobjects	One row for each table, view, procedure, rule, trigger default, log, and (in *tempdb* only) temporary object
syspartitions	One row for each partition (page chain) of a partitioned table
sysprocedures	One row for each view, rule, default, trigger, and procedure, giving internal definition

System Table	Contents
sysprotects	User permissions information
sysreferences	One row for each referential integrity constraint declared on a table or column
sysroles	Maps server-wide roles to local database groups
syssegments	One row for each segment (named collection of disk pieces)
systhresholds	One row for each threshold defined for the database
systypes	One row for each system-supplied and user-defined datatype
sysusermessages	One row for each user-defined message
sysusers	One row for each user allowed in the database

The following system tables occur in the *master* database only:

System Table	Contents
syscharsets	One row for each character set or sort order
sysconfigures	One row for each configuration parameter that can be set by users
syscurconfigs	Information about configuration parameters currently being used by Adaptive Server
sysdatabases	One row for each database on Adaptive Server
sysdevices	One row for each tape dump device, disk dump device, disk for databases, and disk partition for databases
sysengines	One row for each Adaptive Server engine currently online
syslanguages	One row for each language (except U.S. English) known to the server
syslisteners	One row for each type of network connection used by current Adaptive Server
syslocks	Information about active locks
sysloginroles	One row for each server login that possesses a system role
syslogins	One row for each valid Adaptive Server user account
syslogshold	Information about the oldest active transaction and the Replication Server® truncation point for each database

System Table	Contents
sysmessages	One row for each system error or warning
sysmonitors	One row for each monitor counter
sysprocesses	Information about server processes
sysremotelogins	One row for each remote user
sysresourcelimits	One row for each resource limit
syssecmechs	Information about the security services available for each security mechanism that is available to Adaptive Server
sysservers	One row for each remote Adaptive Server
syssrvroles	One row for each server-wide role
systimeranges	One row for each named time range
sysusages	One row for each disk piece allocated to a database

The following system tables occur in the *sybsecurity* database only:

System Table	Contents
sysauditoptions	One row for each global audit option
sysaudits_01, sysaudits_02...sysaudits_08	The audit trail. Each audit table contains one row for each audit record.

In the pages that follow, each system table is described in more detail, including a list of its columns and datatypes, as well as the indexes and Sybase-supplied procedures that reference a particular table are listed.

The word "reserved" in the column description means that the column is not currently used by Adaptive Server.

Note that aggregate functions cannot be used on virtual tables such as *syslocks* and *sysprocesses*.

Permissions on System Tables

Permissions for use of the system tables can be controlled by the database owner, just like permissions on any other tables. By default, when Adaptive Server is installed, the *installmodel* script grants **select** access to "public" (all users) for most system tables and for most fields in the tables. However, no access is given for some system tables, such as *systhresholds*, and no access is given for certain fields in other system tables. For example, all users, by default, can select all columns of *sysobjects* except *audflags*. To determine the current permissions for a particular system table, execute:

```
sp_helprotect system_table_name
```

For example, to check the permissions of *systhresholds* in *your_database*, execute:

```
    use your_database
    go
    sp_helprotect systhresholds
    go
```

Updating System Tables

All direct updates on system tables are by default not allowed — even for the database owner. Instead, Adaptive Server supplies system procedures to make any normally needed updates and additions to system tables.

You can allow direct updates to the system tables if it becomes necessary to modify them in a way that cannot be accomplished with a system procedure. To accomplish this, a System Security Officer must reset the configuration parameter called **allow updates to system tables** with the system procedure **sp_configure**. For information, see the *System Administration Guide*.

There are entries in some of the *master* database tables that should not be altered by any user under any circumstances. For example, do not attempt to modify *syslogs* with a **delete**, **update**, or **insert** command. In addition, an attempt to delete all rows from *syslogs* will put Adaptive Server into an infinite loop that eventually fills up the entire database.

Triggers on System Tables

You cannot create triggers on system tables. If you try to create a trigger on a system table, Adaptive Server returns an error message and cancels the trigger.

sysalternates

(all databases)

Description

sysalternates contains one row for each Adaptive Server user mapped (or aliased) to a user of the current database. When a user tries to access a database, Adaptive Server looks for a valid *uid* entry in *sysusers*. If none is found, it looks in *sysalternates.suid*. If the user's *suid* is found there, he or she is treated as the database user whose *suid* is listed in *sysalternates.altsuid*.

On the Adaptive Server distribution media, there are no entries in *sysalternates*.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID of user being mapped
altsuid	*smallint*	Server user ID of user to whom another user is mapped

Indexes

Unique clustered index on *suid*

Referenced by System Procedures

sp_addalias, **sp_adduser**, **sp_changedbowner**, **sp_dropalias**, **sp_dropuser**, **sp_helpuser**

sysattributes

(all databases)

Description

System attributes define properties of objects such as databases, tables, indexes, users, logins, and procedures. *sysattributes* contains one row for each of an object's attribute definitions (configured by various system procedures). *master..sysattributes* defines the complete set of valid attribute values and classes for Adaptive Server as a whole. It also stores attribute definitions for server-wide objects, such as databases and logins.

sysattributes should only be accessed indirectly using system procedures. The permissions required for modifying *sysattributes* depend on the system procedure you use.

Columns

Name	Datatype	Description
class	*smallint*	The attribute class ID. This describes the category of the attribute. In *master..sysattributes*, the special class 1 identifies all valid attributes for Adaptive Server. Class 0 identifies valid **classes** of attributes.
attribute	*smallint*	The attribute ID.
object_type	*char(2)*	The one- or two-letter character ID that defines the type of object to associate with the attribute.
object_cinfo	*varchar(30)*	A string identifier for the object (for example, the name of an application). This field is not used by all attributes.
object	*int null*	The object identifier. This may be an object ID, user ID, or database ID, depending on the type of object. If the object is a part of a table (for example, an index), then this column contains the object ID of the associated table.
object_info1	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.

Name	Datatype	Description
object_info2	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.
object_info3	*int null*	Defines additional information required to identify the object. This field is not used by all attributes. The contents of this field depend on the attribute that is defined.
int_value	*int null*	An integer value for the attribute (for example, the display level of a user).
char_value	*varchar(255)*	A character value for the attribute (for example, a cache name).
text_value	*text null*	A text value for the attribute.
image_value	*image null*	An image value for the attribute.
comments	*varchar(255)*	Comments or additional information about the attribute definition.

Table 8-1 describes the *object_type* values and their meanings:

Table 8-1: Object types for attributes

ID	Object Type
D	Database
EL	External Login (for Component Integration Services)
I	Index
L	Login name
OD	Object Definition (for Component Integration Services)
P	Procedure
T	Table
TP	Text Page (for Component Integration Services)
U	Username
UI	Upgrade Item (used internally during user database upgrades)

Indexes

Unique clustered index on *class*, *attribute*, *object_type*, *object*, *object_info1*, *object_info2*, *object_info3*, *object_cinfo*

Nonclustered index on *object_type*, *object*, *object_info1*, *object_info2*, *object_info3*, *object_cinfo*

Referenced by System Procedures

sp_activeroles, sp_addengine, sp_addexeclass, sp_addexternlogin, sp_addobjectdef, sp_bindcache, sp_bindexeclass, sp_clearpsexe, sp_configure, sp_displaylevel, sp_displayroles, sp_dropengine, sp_dropexeclass, sp_dropexternlogin, sp_dropglockpromote, sp_droplogin, sp_dropobjectdef, sp_dropserver, sp_dropuser, sp_forceonline_db, sp_forceonline_page, sp_help, sp_helpdb, sp_helpexternlogin, sp_helpindex, sp_helpobjectdef, sp_helprotect, sp_listsuspect_db, sp_listsuspect_page, sp_logiosize, sp_setpglockpromote, sp_setpsexe,, sp_setsuspect_granularity, sp_setsuspect_threshold, sp_shmdumpconfig, sp_showcontrolinfo, sp_showexeclass, sp_unbindexeclass

sysauditoptions

(*sybsecurity* database)

Description

sysauditoptions contains one row for each server-wide audit option and indicates the current setting for that option. Other types of auditing option settings are stored in other tables. For example, database-specific option settings are stored in *sysdatabases*, and object-specific option settings are stored in *sysobjects*. The default value for each option is 0, or "off." *sysauditoptions* can be accessed only by System Security Officers.

Columns

Name	Datatype	Description
num	*smallint*	Number of the server-wide option.
val	*smallint*	Current value; one of the following: 0 = **off** 1 = **pass** 2 = **fail** 3 = **on**
minval	*smallint*	Minimum valid value for this option.
maxval	*smallint*	Maximum valid value for this option.
name	*varchar(30)*	Name of option.
sval	*varchar(30)*	String equivalent of the current value: for example, "on", "off", "nonfatal".
comment	*varchar(255)*	Description of option.

Indexes

None

Referenced by System Procedures

sp_addauditrecord, **sp_audit**

sysaudits_01, sysaudits_02...sysaudits_08

(*sybsecurity* database)

Description

These system tables contain the audit trail. Only one table at a time is active. The active table is determined by the value of the **current audit table** configuration parameter. An installation can have up to eight audit tables. For example, if your installation has three audit tables, the tables are named *sysaudits_01*, *sysaudits_02*, and *sysaudits_03*. An audit table contains one row for each audit record.

Columns

Name	Datatype	Description
event	*smallint*	Type of event being audited. See Table 8-3 on page 8-12.
eventmod	*smallint*	Further information about the event. Possible values are: 0 = no modifier for this event 1 = the event passed permission checking 2 = the event failed permission checking
spid	*smallint*	Server process ID of the process that caused the audit record to be written.
eventtime	*datetime*	Date and time of the audited event.
sequence	*smallint*	Sequence number of the record within a single event; some events require more than one audit record.
suid	*smallint*	Server login ID of the user who performed the audited event.
dbid	*int null*	Database ID in which the audited event occurred or the object / stored procedure / trigger resides, depending on the type of event.
objid	*int null*	ID of the accessed object or stored procedure / trigger.
xactid	*binary(6) null*	ID of the transaction containing the audited event. For a multi-database transaction, this is the transaction ID from the database where the transaction originated.
loginname	*varchar(30) null*	Login name corresponding to the *suid.*
dbname	*varchar(30) null*	Database name corresponding to the *dbid.*

Name	Datatype	Description
objname	*varchar(30) null*	Object name corresponding to the *objid.*
objowner	*varchar(30) null*	Name of the owner of *objid.*
extrainfo	*varchar(255) null*	Additional information about the audited event. This field contains a sequence of items separated by semicolons. See Table 8-2.

The *extrainfo* column contains a sequence of items separated by semicolons. Table 8-2 lists the items in the *extrainfo* column:

Table 8-2: Items in the extrainfo field

Item	Contents
Roles	Lists the roles that are active. The roles are separated by blanks.
Subcommand	The name of the subcommand or command option that was used for the event. For example, for the **alter table** command, the options "add column" or "drop constraint" might be used. Multiple subcommands or options are separated by commas.
Previous value	The value prior to the update if the event resulted in the update of a value.
Current value	The new value if the event resulted in the update of a value.
Other information	Additional security-relevant information that is recorded for the event.
Proxy information	The original login name, if the event occurred while a **set proxy** was in effect.
Principal information	The principal name from the underlying security mechanism, if the user's login is the secure default login, and the user logged into Adaptive Server via unified login. The value of this field is NULL, if the secure default login is not being used.

An example of an *extrainfo* column for the security-relevant event of changing an auditing configuration parameter might be:

```
sso_role;suspend auditing when full;1;0;;;;
```

This *extrainfo* column indicates that a System Security Officer changed the configuration parameter **suspend auditing when full** from 1 (suspend all processes that involve an auditing event) to 0 (truncate the next audit table and make it the current audit table). The other columns in the audit record give other pertinent information. For example, the record contains the server user id (*suid*) and the login name (*loginname*).

The *event* column values that pertain to each audit event are listed in Table 8-3.

Table 8-3: Values in event and extrainfo column

Audit Option	*event*	Command or Access Audited	*extrainfo*
adhoc	1	User-defined audit record	*extrainfo* is filled by the *text* parameter of **sp_addauditrecord**
alter	2	**alter database**	**Roles:** Current active roles **Subcommand:** “ALTER SIZE” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	3	**alter table**	**Roles:** Current active roles **Subcommand:** “ADD COLUMN”, “REPLACE COLUMN”, “ADD CONSTRAINT”, or “DROP CONSTRAINT” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
bcp	4	**bcp in**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
bind	6	**sp_bindefault**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Name of default **Proxy information:** Original login name, if a **set proxy** is in effect
	7	**sp_bindmsg**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Message ID **Proxy information:** Original login name, if a **set proxy** is in effect
	8	**sp_bindrule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Name of the rule **Proxy information:** Original login name, if a **set proxy** is in effect
create	9	**create database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	10	**create table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	11	**create procedure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
create (continued)	12	**create trigger**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	13	**create rule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	14	**create default**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	15	**sp_addmessage**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Message Number **Proxy information:** Original login name, if a **set proxy** is in effect
	16	**create view**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
dbaccess	17	Any access to the database by any user	**Roles:** Current active roles **Subcommand:** "USE CMD" or "OUTSIDE REFERENCE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
dbcc	81	**dbcc**	**Roles:** Current active roles **Subcommand:** The **dbcc** subcommand name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
delete	18	**delete** from a table	**Roles:** Current active roles **Subcommand:** “DELETE” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	19	**delete** from a view	**Roles:** Current active roles **Subcommand:** “DELETE” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
disk	20	**disk init**	**Roles:** Current active roles **Subcommand:** “disk init” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	21	**disk refit**	**Roles:** Current active roles **Subcommand:** “disk refit” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
disk (continued)	22	**disk reinit**	**Roles:** Current active roles **Subcommand:** “disk reinit” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	23	**disk mirror**	**Roles:** Current active roles **Subcommand:** “disk mirror” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	24	**disk unmirror**	**Roles:** Current active roles **Subcommand:** “disk unmirror” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
	25	**disk remirror**	**Roles:** Current active roles **Subcommand:** “disk remirror” **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the disk **Proxy information:** Original login name, if a **set proxy** is in effect
drop	26	**drop database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	27	**drop table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
drop (continued)	28	**drop procedure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	29	**drop trigger**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	30	**drop rule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	31	**drop default**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	32	**sp_dropmessage**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Message number **Proxy information:** Original login name, if a **set proxy** is in effect
	33	**drop view**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
dump	34	**dump database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	35	**dump transaction**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
errors	36	Fatal error	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Error number.Severity.State **Proxy information:** Original login name, if a **set proxy** is in effect
	37	Non-fatal error	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** Error number.Severity.State **Proxy information:** Original login name, if a **set proxy** is in effect
exec_procedure	38	Execution of a procedure	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** All input parameters **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
exec_trigger	39	Execution of a trigger	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
func_obj_access, func_dbaccess	85	Accesses to objects and databases via Transact-SQL functions	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
grant	40	**grant**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
insert	41	**insert** into a table	**Roles:** Current active roles **Subcommand:** If **insert**: “INSERT” If **select into**: “INSERT INTO” followed by the fully qualified object name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	42	**insert** into a view	**Roles:** Current active roles **Subcommand:** “INSERT” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
load	43	**load database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	44	**load transaction**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
login	45	Any login to Adaptive Server	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Host name of the machine from which login was done **Proxy information:** Original login name, if a **set proxy** is in effect
logout	46	Any logouts from Adaptive Server	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Host name of the machine from which login was done **Proxy information:** Original login name, if a **set proxy** is in effect
revoke	47	**revoke**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
rpc	48	Remote procedure call from another server	**Roles:** Current active roles **Subcommand:** Name of client program **Previous value:** NULL **Current value:** NULL **Other information:** Server name, host name of the machine from which the RPC was done. **Proxy information:** Original login name, if a **set proxy** is in effect
	49	Remote procedure call to another server	**Roles:** Current active roles **Subcommand:** Procedure name **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
security	50	Server start	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** **-d***masterdevicename* **-i***interfaces file path* **-S***servername* **-e***errorfilename* **Proxy information:** Original login name, if a **set proxy** is in effect
	51	Server shutdown	**Roles:** Current active roles **Subcommand:** “shutdown” **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	55	Role toggling	**Roles:** Current active roles **Subcommand:** NULL **Previous Value:** “on” or “off” **Current Value:** “on” or “off” **Other Information:** Name of the role being set **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
security (continued)	82	**sp_configure**	**Roles:** Current active roles **Subcommand:** Name of the configuration parameter **Previous Value:** The old parameter value if the command is setting a new value **Current Value:** The new parameter value if the command is setting a new value **Other Information:** Number of configuration parameter, if a parameter is being set; Name of the configuration file, if a configuration file is being used to set parameters **Proxy information:** Original login name, if a **set proxy** is in effect
	83	**online database**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	76	Regeneration of a password by a System Security Officer (SSO)	**Roles:** Current active roles **Subcommand:** Setting SSO password **Previous value:** NULL **Current value:** NULL **Other information:** Login name **Proxy information:** Original login name, if a **set proxy** is in effect
	80	**proc_role** within a system procedure	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Required roles **Proxy information:** Original login name, if a **set proxy** is in effect
	85	**valid_user**	**Roles:** Current active roles **Subcommand:** "valid_user" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
security (continued)	88	**set proxy** or **set session authorization**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** Previous *suid* **Current value:** New *suid* **Other information:** NULL **Proxy information**: Orginal login name, if **set proxy** or **set session authorization** had no parameters; otherwise, NULL.
select	62	**select** from a table	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	63	**select** from a view	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
setuser	84	**setuser**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other Information:** Name of the user being set **Proxy information:** Original login name, if a **set proxy** is in effect
table_access	62	**select**	**Roles:** Current active roles **Subcommand:** "SELECT INTO", "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
table_access (continued)	18	**delete**	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	70	**update**	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	41	**insert**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
truncate	64	**truncate table**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
unbind	67	**sp_unbindefault**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	68	**sp_unbindrule**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
unbind (continued)	69	**sp_unbindmsg**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
update	70	**update** to a table	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	71	**update** to a view	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
view_access	63	**select**	**Roles:** Current active roles **Subcommand:** "SELECT INTO" "SELECT", or "READTEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	19	**delete**	**Roles:** Current active roles **Subcommand:** "DELETE" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Table 8-3: Values in event and extrainfo column (continued)

Audit Option	*event*	Command or Access Audited	*extrainfo*
view_access (continued)	42	**insert**	**Roles:** Current active roles **Subcommand:** "INSERT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
	71	**update**	**Roles:** Current active roles **Subcommand:** "UPDATE" or "WRITETEXT" **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
Note: This event is audited automatically. It is not controlled by an audit option.	73	Turning the **auditing** parameter on with **sp_configure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect
Note: This event is audited automatically. It is not controlled by an audit option.	74	Turning the **auditing** parameter off with **sp_configure**	**Roles:** Current active roles **Subcommand:** NULL **Previous value:** NULL **Current value:** NULL **Other information:** NULL **Proxy information:** Original login name, if a **set proxy** is in effect

Indexes

None

Referenced by System Procedures

sp_addauditrecord, **sp_audit**

syscharsets

(*master* database only)

Description

syscharsets contains one row for each character set and sort order defined for use by Adaptive Server. One of the sort orders is marked in *master..sysconfigures* as the default sort order, which is the only one actually in use.

Columns

Name	Datatype	Description
type	*smallint*	The type of entity this row represents. Numbers from 1001 to 1999 represent character sets. Numbers from 2000 to 2999 represent sort orders.
id	*tinyint*	The ID for a character set or sort order. A sort order is defined by the combination of the sort order ID and the character set ID (*csid*). The character set is defined by *id*, which must be unique. Sybase reserves ID numbers 0–200.
csid	*tinyint*	If the row represents a character set, this field is unused. If the row represents a sort order, this is the ID of the character set that sort order is built on. A character set row with this ID must exist in this table.
status	*smallint*	Internal system status information bits.
name	*varchar(30)*	A unique name for the character set or sort order. Must contain only the 7-bit ASCII letters A-Z or a-z, digits 0-9, and underscores (_), and begin with a letter.
description	*varchar(255)*	An optional description of the features of the character set or sort order.
definition	*image*	The internal definition of the character set or sort order. The structure of the data in this field depends on the *type*.

Indexes

Unique clustered index on *id, csid, type*
Unique nonclustered index on *name*

Referenced by System Procedures

sp_checkreswords, sp_configure, sp_helpsort, sp_serverinfo

syscolumns

(all databases)

Description

syscolumns contains one row for every column in every table and view, and a row for each parameter in a procedure.

Columns

Name	Datatype	Description
id	*int*	ID of table to which this column belongs or of procedure with which this parameter is associated
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
colid	*tinyint*	Column ID
status	*tinyint*	Indicates unique position for *bit* columns, whether NULL values are legal in this column, and if more than one check constraint exists for the column
type	*tinyint*	Physical storage type; copied from *systypes*
length	*tinyint*	Physical length of data; copied from *systypes* or supplied by user
offset	*smallint*	Offset into the row where this column appears; if negative, this is a variable-length column
usertype	*smallint*	User type ID; copied from *systypes*
cdefault	*int*	ID of the procedure that generates default value for this column
domain	*int*	Constraint ID of the first rule or check constraint for this column
name	*sysname*	Column name
printfmt	*varchar(255)*	Reserved
prec	*tinyint*	Number of significant digits
scale	*tinyint*	Number of digits to the right of the decimal point

Name	Datatype	Description
remote_type	*int*	Maps local names to remote names. Required by the access methods of Component Integration Services to allow the software to pass native column datatype information in parameters to servers of class *access_server*.
remote_name	*varchar(30)*	Maps local names to remote names. Required by the access methods of Component Integration Services to construct a query using the proper column names for a remote table.

Indexes

Unique clustered index on *id, number, colid*

Referenced by System Procedures

sp_bindefault, sp_bindrule, sp_changegroup, sp_checkreswords, sp_column_privileges, sp_commonkey, sp_droptype, sp_dropuser, sp_estspace, sp_foreignkey, sp_help, sp_helpjoins, sp_helprotect, sp_primarykey, sp_rename, sp_statistics, sp_unbindefault, sp_unbindrule

syscomments

(all databases)

Description

syscomments contains entries for each view, rule, default, trigger, table constraint, and procedure. The *text* column contains the original definition statements. If the *text* column is longer than 255 bytes, the entries will span rows. Each object can occupy up to 65,025 rows.

Columns

Name	Datatype	Description
id	*int*	Object ID to which this text applies
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
colid	*tinyint*	Sequence of 255 rows for the object
texttype	*smallint*	0 for system-supplied comment (for views, rules, defaults, triggers, and procedures); 1 for user-supplied comment (users can add entries that describe an object or column)
language	*smallint*	Reserved
text	*varchar(255)*	Actual text of SQL definition statement
colid2	*tinyint*	Indicates next sequence of rows for the object (see *colid* above); object can have up to 255 sequences of 255 rows each
status	*smallint*	

➤ ***Note***

Do not delete the definition statements from the text column of *syscomments*. These statements are required for the Adaptive Server upgrade process. To encrypt a definition statement, run the system procedure **sp_hidetext**. To see if a statement created in release 11.5 or later was deleted, run **sp_checksource**. If the statement was deleted, you must either recreate the object that created the statement or reinstall the application that created the object, which will re-create the statement.

You can protect the text of a database object against unauthorized access by restricting **select** permission on the *text* column of the *syscomments* table to the owner of the object and the System Administrator. This restriction, which applies to direct access through **select** statements as well as access through stored procedures, is required in order to run Adaptive Server in the evaluated configuration. To enact this restriction, a System Security Officer must reset the parameter called **allow select on syscomments.text column** with the system procedure **sp_configure**. For information, see the *System Administration Guide*.

Indexes

Unique clustered index on *id, number, colid2, colid, texttype*

Referenced by System Procedures

sp_checksource, **sp_helpconstraint**, **sp_helptext**, **sp_hidetext**

sysconfigures

(*master* database only)

Description

sysconfigures contains one row for each configuration parameter that can be set by the user.

Columns

Name	Datatype	Description
config	*smallint*	Configuration parameter number.
value	*int*	The user-modifiable value for the parameter with *integer* datatype. Its value is 0 for the parameters with *character* datatype.
comment	*varchar(255)*	Name of the configuration parameter.
status	*smallint*	Either 1 (dynamic) or 0 (parameter takes effect when Adaptive Server is restarted).
name	*varchar(80)*	Name of the configuration parameter (the same value as *comment*).
parent	*smallint*	Configuration parameter number of the parent; if more than one parent, the additional parent numbers are stored in *sysattributes*.
value2	*varchar(255)*	The user-modified value for the parameter with the character datatype. Its value is NULL for parameters with *integer* datatype. It is also used to store the pool size of a buffer pool.
value3	*int*	Stores the wash size of a buffer pool.
value4	*int*	Stores the asynchronous prefetch percents of a buffer pool.

Indexes

Unique clustered index on *config, name, parent*
Nonclustered index on *config, parent*
Nonclustered index on *config*

Referenced by System Procedures

sp_configure

sysconstraints

(all databases)

Description

The *sysconstraints* table has one row for each referential constraint and check constraint associated with a table or column.

Whenever a user declares a new check constraint or referential constraint using **create table** or **alter table**, Adaptive Server inserts a row into the *sysconstraints* table. The row remains until a user executes **alter table** to drop the constraint. Dropping a table by executing **drop table** removes all rows associated with that table from the *sysconstraints* table.

Columns

Name	Datatype	Description
colid	*tinyint*	Column number in the table
spare1	*tinyint*	Unused
constrid	*int*	Object ID of the constraint
tableid	*int*	ID of the table on which the constraint is declared
error	*int*	Constraint specific error message
status	*int*	The type of constraint: 0x0040 = a referential constraint 0x0080 = a check constraint
spare2	*int*	Unused

Indexes

Clustered index on *tableid, colid*
Unique nonclustered index on *constrid*

Referenced by System Procedures

sp_bindmsg, sp_bindrule, sp_helpconstraint, sp_unbindmsg, sp_unbindrule

syscurconfigs

(*master* database only)

Description

syscurconfigs is built dynamically when queried. It contains an entry for each of the configuration parameters, as does *sysconfigures*, but with the current values rather than the default values. In addition, it contains four rows that describe the configuration structure.

Columns

Name	Datatype	Description
config	*smallint*	Configuration parameter number.
value	*int*	The current run value for the parameter with *integer* datatype. Its value is 0 for the parameters with character datatype.
comment	*varchar(255)*	Amount of memory used by each configuration parameter, represented in a string format. Values marked with a hash mark (#) share memory with other parameters.
status	*smallint*	Either 1 (dynamic) or 0 (parameter takes effect when Adaptive Server is restarted).
value2	*varchar(255)*	The current run value for the parameter with the *character* datatype. Its value is NULL for parameters with the *integer* datatype.
defvalue	*varchar(255)*	Default value of the configuration parameter.
minimum_value	*int*	Minimum value of the configuration parameter.
maximum_value	*int*	Maximum value of the configuration parameter.
memory_used	*int*	Integer value for the amount of memory used by each configuration parameter.
display_level	*int*	Display level of the configuration parameter (the values are 1, 5, and 10).
datatype	*int*	Datatype of the configuration parameter.
message_num	*int*	Unused.
apf_percent	*int*	The current run value for the asynchronous prefetch percent for a buffer pool. Valid only for rows that represent buffer pools.

Indexes

None

Referenced by System Procedures

sp_configure, sp_countmetadata, sp_helpconfig, sp_helpserver, sp_helpsort, sp_helptext, sp_procqmode, sp_serverinfo

sysdatabases

(*master* database only)

Description

sysdatabases contains one row for each database in Adaptive Server. When Adaptive Server is installed, *sysdatabases* contains entries for the *master* database, the *model* database, the *sybsystemprocs* database, and the *tempdb* database. If you have installed auditing, it also contains an entry for the *sybsecurity* database.

Columns

Name	Datatype	Description
name	*sysname*	Name of the database
dbid	*smallint*	Database ID
suid	*smallint*	Server user ID of database owner
status	*smallint*	Control bits; those that the user can set with **sp_dboption** are so indicated in Table 8-4
version	*smallint*	Unused
logptr	*int*	Pointer to transaction log
crdate	*datetime*	Creation date
dumptrdate	*datetime*	Date of the last **dump transaction**
status2	*intn*	Additional control bits (see Table 8-5)
audflags	*intn*	Audit settings for database
deftabaud	*intn*	Bit-mask that defines default audit settings for tables
defvwaud	*intn*	Bit-mask that defines default audit settings for views
defpraud	*intn*	Bit-mask that defines default audit settings for stored procedures
def_remote_type	*smallint*	Identifies the default object type to be used for remote tables if no storage location is provided via the stored procedure **sp_addobjectdef**

Name	Datatype	Description
def_remote_loc	*varchar(255)*	Identifies the default storage location to be used for remote tables if no storage location is provided via the stored procedure **sp_addobjectdef**

Table 8-4 lists the bit representations for the *status* column.

Table 8-4: status control bits in the sysdatabases table

Decimal	Hex	Status
4	0x04	**select into/bulkcopy**; can be set by user
8	0x08	**trunc log on chkpt**; can be set by user
16	0x10	**no chkpt on recovery**; can be set by user
32	0x20	Database created with **for load** option, or crashed while loading database, instructs recovery not to proceed
256	0x100	Database suspect; not recovered; cannot be opened or used; can be dropped only with **dbcc dbrepair**
512	0x200	**ddl in tran**; can be set by user
1024	0x400	**read only**; can be set by user
2048	0x800	**dbo use only**; can be set by user
4096	0x1000	**single user**; can be set by user
8192	0x2000	**allow nulls by default**; can be set by user

Table 8-5 lists the bit representations for the *status2* column.

Table 8-5: status2 control bits in the sysdatabases table

Decimal	Hex	Status
1	0x0001	**abort tran on log full**; can be set by user
2	0x0002	**no free space acctg**; can be set by user
4	0x0004	**auto identity**; can be set by user
8	0x0008	**identity in nonunique index**; can be set by user
16	0x0010	Database is offline
32	0x0020	Database is offline until recovery completes
64	0x0040	Database is being recovered (internal use)
32768	0x8000	Database does not have a dedicated log device

Indexes

Unique clustered index on *name*
Unique nonclustered index on *dbid*

Referenced by System Procedures

sp_addlogin, **sp_addsegment**, **sp_addtype**, **sp_audit**, **sp_changedbowner**, **sp_checknames**, **sp_checkreswords**, **sp_databases**, **sp_dboption**, **sp_dbremap**,

sp_dropdevice, sp_dropsegment, sp_extendsegment, sp_helpdb, sp_logdevice, sp_renamedb, sp_tables

sysdepends

(all databases)

Description

sysdepends contains one row for each procedure, view, or table that is referenced by a procedure, view, or trigger.

Columns

Name	Datatype	Description
id	*int*	Object ID
number	*smallint*	Procedure number
depid	*int*	Dependent object ID
depnumber	*smallint*	Dependent procedure number
status	*smallint*	Internal status information
selall	*bit*	On if object is used in **select** * statement
resultobj	*bit*	On if object is being updated
readobj	*bit*	On if object is being read

Indexes

Unique clustered index on *id, number, depid, depnumber*

Referenced by System Procedures

sp_audit, **sp_depends**

sysdevices

(*master* database only)

Description

sysdevices contains one row for each tape dump device, disk dump device, disk for databases, and disk partition for databases. On the Adaptive Server distribution media, there are four entries in *sysdevices*: one for the master device (for databases), one for a disk dump device, and two for tape dump devices.

Columns

Name	Datatype	Description
low	*int*	First virtual page number on database device (not used for dump devices)
high	*int*	Last virtual page number on database device or dump device
status	*smallint*	Bitmap indicating type of device, default and mirror status (see Table 8-6)
cntrltype	*smallint*	Controller type (0 if database device, 2 if disk dump device or streaming tape, 3–8 if tape dump device)
name	*sysname*	Logical name of dump device or database device
phyname	*varchar(127)*	Name of physical device
mirrorname	*varchar(127)*	Name of mirror device

The bit representations for the *status* column, shown in Table 8-6, are additive. For example, "3" indicates a physical disk that is also a default.

Table 8-6: status control bits in the sysdevices table

Decimal	Hex	Status
1	0x01	Default disk
2	0x02	Physical disk
4	0x04	Logical disk (not used)
8	0x08	Skip header
16	0x10	Dump device
32	0x20	Serial writes
64	0x40	Device mirrored

Table 8-6: status control bits in the sysdevices table (continued)

Decimal	Hex	Status
128	0x80	Reads mirrored
256	0x100	Secondary mirror side only
512	0x200	Mirror enabled
1024	0x400	Master device is mirrored
2048	0x800	Mirror disabled (used internally)
4096	0x1000	Primary device needs to be unmirrored (used internally)
8192	0x2000	Secondary device needs to be unmirrored (used internally)

Indexes

Unique clustered index on *name*

Referenced by System Procedures

sp_addsegment, **sp_addumpdevice**, **sp_checknames**, **sp_checkreswords**, **sp_configure**, **sp_diskdefault**, **sp_dropdevice**, **sp_dropsegment**, **sp_extendsegment**, **sp_helpdb**, **sp_helpdevice**, **sp_helplog**, **sp_helpsegment**, **sp_logdevice**, **sp_volchanged**

sysengines

(*master* database only)

Description

sysengines contains one row for each Adaptive Server engine currently online.

Columns

Name	Datatype	Description
engine	*smallint*	Engine number
osprocid	*int*	Operating system process ID (may be NULL)
osprocname	*char*	Operating system process name (may be NULL)
status	*char*	One of: online, offline, in create, in destroy, debug
affinitied	*int*	Number of Adaptive Server processes with affinity to this engine
cur_kpid	*int*	Kernel process ID of process currently running on this engine, if any
last_kpid	*int*	Kernel process ID of process that previously ran on this engine
idle_1	*tinyint*	Reserved
idle_2	*tinyint*	Reserved
idle_3	*tinyint*	Reserved
idle_4	*tinyint*	Reserved
starttime	*datetime*	Date and time engine came online

Indexes

None

Referenced by System Procedures

sp_monitor

sysgams

(all databases)

Description

sysgams stores the global allocation map (GAM) for the database. The GAM stores a bitmap for all allocation units of a database, with one bit per allocation unit. You cannot select from or view *sysgams*.

Columns

None

Indexes

None

Referenced by System Procedures

None

sysindexes

(all databases)

Description

sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains *text* or *image* columns.

Columns

Name	Datatype	Description
name	*sysname*	Index or table name
id	*int*	ID of table, or ID of table to which the index belongs
indid	*smallint*	0 if table, 1 if clustered index, >1 if nonclustered, 255 if text chain
doampg	*int*	Page number for the object allocation map of a table or clustered index
ioampg	*int*	Page number for the allocation map of a nonclustered index
oampgtrips	*int*	Ratio of OAM page to data page residency in cache
status2	*int*	Internal system status information (see Table 8-7)
ipgtrips	*int*	Ratio of index page to data page residency in cache
first	*int*	Page number of the first data or leaf page
root	*int*	Page number of the root page if entry is an index; page number of the last page if entry is an unpartitioned table or text chain; unused if entry is a partitioned table (see *syspartitions*)
distribution	*int*	Page number of the distribution page (if entry is an index)
usagecnt	*smallint*	Reserved
segment	*smallint*	Number of segment in which this object resides
status	*smallint*	Internal system status information (see Table 8-8)

Name	Datatype	Description
maxrowsperpage	*smallint*	Maximum number of rows per page
minlen	*smallint*	Minimum size of a row
maxlen	*smallint*	Maximum size of a row
maxirow	*smallint*	Maximum size of a non-leaf index row
keycnt	*smallint*	Number of keys for a clustered index; number of keys+1 for a nonclustered index
keys1	*varbinary(255)*	Description of key columns (if entry is an index)
keys2	*varbinary(255)*	Description of key columns (if entry is an index)
soid	*tinyint*	Sort order ID that the index was created with; 0 if there is no character data in the keys
csid	*tinyint*	Character set ID that the index was created with; 0 if there is no character data in the keys
base_partition	*int*	Partition number, incremented by **alter table...unpartition** commands

The *doampg* column is used only if the row describes a table or clustered index; this column and the *ioampg* column are used by the system functions **data_pgs**, **reserved_pgs,** and **used_pgs**.

Table 8-7 lists the bit representations for the *status2* column.

Table 8-7: status2 control bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Index supports foreign key constraint
2	0x2	Index supports primary key / unique declarative constraint
4	0x4	Index includes an IDENTITY column
8	0x8	User did not specify a constraint name
16	0x10	Large I / Os (prefetch) enabled for table, index, or text chain
32	0x20	MRU cache strategy enabled for table, index, or text chain
64	0x40	Ascending inserts turned on for the table

Table 8-8 lists the bit representations for the *status* column.

Table 8-8: status control bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Abort current command or trigger if an attempt is made to insert duplicate key
2	0x2	Unique index
4	0x4	Abort current command or trigger if an attempt is made to insert duplicate row
16	0x10	Clustered index
64	0x40	Index allows duplicate rows
128	0x80	Sorted object; not set for tables without clustered indexes or for text objects
512	0x200	**sorted data** option used in **create index** statement
1024	0x400	Index being created
2048	0x800	Index on primary key
32768	0x8000	Suspect index; index was created under another sort order

Indexes

Unique clustered index on *id, indid*

Referenced by System Procedures

sp_cachestrategy, **sp_checknames**, **sp_checkreswords**, **sp_dropsegment**, **sp_estspace**, **sp_help**, **sp_helpconstraint**, **sp_helpindex**, **sp_helplog**, **sp_helpsegment**, **sp_indsuspect**, **sp_pkeys**, **sp_placeobject**, **sp_relimit**, **sp_rename**, **sp_spaceused**, **sp_special_columns**, **sp_statistics**

syskeys

(all databases)

Description

syskeys contains one row for each primary, foreign, or common key.

Columns

Name	Datatype	Description
id	*int*	Object ID
type	*smallint*	Record type
depid	*int null*	Dependent object ID
keycnt	*int null*	Number of non-null keys
size	*int null*	Reserved
key1	*int null*	Column ID
key2	*int null*	Column ID
key3	*int null*	Column ID
key4	*int null*	Column ID
key5	*int null*	Column ID
key6	*int null*	Column ID
key7	*int null*	Column ID
key8	*int null*	Column ID
depkey1	*int null*	Column ID
depkey2	*int null*	Column ID
depkey3	*int null*	Column ID
depkey4	*int null*	Column ID
depkey5	*int null*	Column ID
depkey6	*int null*	Column ID
depkey7	*int null*	Column ID
depkey8	*int null*	Column ID

Indexes

Clustered index on *id*

Referenced by System Procedures

sp_commonkey, sp_dropkey, sp_foreignkey, sp_helpjoins, sp_helpkey, sp_primarykey

syslanguages

(*master* database only)

Description

syslanguages contains one row for each language known to Adaptive Server. us_english is not in *syslanguages*, but it is always available to Adaptive Server.

Colomns

Name	Datatype	Description
langid	*smallint*	Unique language ID
dateformat	*char(3)*	Date order; for example, “dmy”
datefirst	*tinyint*	First day of the week—1 for Monday, 2 for Tuesday, and so on, up to 7 for Sunday
upgrade	*int*	Adaptive Server version of last upgrade for this language
name	*varchar(30)*	Official language name, for example, “french”
alias	*varchar(30)*	Alternate language name, for example, “français”
months	*varchar(251)*	Comma-separated list of full-length month names, in order from January to December—each name is at most 20 characters long
shortmonths	*varchar(119)*	Comma-separated list of shortened month names, in order from January to December—each name is at most 9 characters long
days	*varchar(216)*	Comma-separated list of day names, in order from Monday to Sunday—each name is at most 30 characters long

Indexes

Unique clustered index on *langid*
Unique nonclustered index on *name*
Unique nonclustered index on *alias*

Referenced by System Procedures

sp_addlanguage, **sp_addmessage**, **sp_checkreswords**, **sp_configure**, **sp_droplanguage**, **sp_dropmessage**, **sp_getmessage**, **sp_helplanguage**, **sp_setlangalias**

syslisteners

(*master* database only)

Description

syslisteners contains a row for each network protocol available for connecting with the current Adaptive Server. Adaptive Server builds *syslisteners* dynamically when a user or client application queries the table.

Columns

Name	Datatype	Description
net_type	*char(32)*	Network protocol
address_info	*char(255)*	Information that uniquely identifies this Adaptive Server on the network, usually the name of the current Adaptive Server and an identifying number, such as the server's port number for the protocol

Indexes

None

Referenced by System Procedures

None

syslocks

(*master* database only)

Description

syslocks contains information about active locks, but it is not a normal table. Rather, it is built dynamically when queried by a user. No updates to *syslocks* are allowed.

Columns

Name	Datatype	Description
id	*int*	Table ID
dbid	*smallint*	Database ID
page	*int*	Page number
type	*smallint*	Type of lock (bit values for the *type* column are listed in Table 8-9)
spid	*smallint*	ID of process that holds the lock
class	*char(30)*	Name of the cursor this lock is associated with, if any
fid	*smallint*	The family (coordinating process and its worker processes) to which the lock belongs. *fid* values are listed in Table 8-10.
context	*tinyint*	Context type of lock request. *context* values are listed in Table 8-11.

Table 8-9 lists the bit representations for the *type* column.

Table 8-9: type control bits in the syslocks table

Decimal	Hex	Status
1	0x1	Exclusive table lock
2	0x2	Shared table lock
3	0x3	Exclusive intent lock (will do page locking on indicated pages)
4	0x4	Shared intent lock
5	0x5	Exclusive page lock
6	0x6	Shared page lock
7	0x7	Update page lock (changes to exclusive if page is modified)
256	0x100	Lock is blocking another process
512	0x200	Demand lock

Table 8-10 lists the values for the *fid* column:

Table 8-10: fid column values in the syslocks table

Value	Interpretation
0	The task represented by the *spid* is a single task executing a statement in serial.
Nonzero value	The task (*spid*) holding the lock is a member of a family executing a statement in parallel. If the value is equal to the *spid*, it indicates that the task is the coordinating process in a family executing a query in parallel.

Table 8-11 lists the values for the *context* column:

Table 8-11: context column values in the syslocks table

Value	Interpretation
Null	The task holding this lock is either executing a query in serial, or it is a query being executed in parallel in transaction isolation level 1.
0x1	The task holding the lock will hold the lock until the query is complete. A lock's context may be FAM_DUR (0x1H) under the following conditions: • The lock is a table lock held as part of a parallel query • The lock is held by a worker process at transaction isolation level 3 • The lock is held by a worker process in a parallel query and must be held for the duration of the transaction

Indexes

None

Referenced by System Procedures

sp_familylock, **sp_lock**

sysloginroles

(master **database only)**

Description

sysloginroles contains a row for each instance of a server login possessing a system role. One row is added for each role granted to each login. For example, if a single server user is granted **sa_role**, **sso_role**, and **oper_role**, three rows are added to *sysloginroles* associated with that user's system user ID (*suid).*

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID
srid	*smallint*	Server role ID; one of the following: 0 = **sa_role** 1 = **sso_role** 2 = **oper_role** 4 = **navigator_role** 5 = **replication_role**
status	*smallint*	Reserved

Indexes

Clustered index on *suid*

Referenced by System Procedures

sp_displaylogin, **sp_droplogin**, **sp _locklogin**, **sp_role**

syslogins

(*master* database only)

Description

syslogins contains one row for each valid Adaptive Server user account.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID
status	*smallint*	Status of the account (see Table 8-12)
accdate	*datetime*	Date *totcpu* and *totio* were last cleared
totcpu	*int*	CPU time accumulated by login
totio	*int*	I/O accumulated by login
spacelimit	*int*	Reserved
timelimit	*int*	Reserved
resultlimit	*int*	Reserved
dbname	*sysname*	Name of database in which to put user when connection established
name	*sysname*	Login name of user
password	*varbinary*	Password of user (encrypted)
language	*varchar(30)*	User's default language
pwdate	*datetime*	Date the password was last changed
audflags	*int*	User's audit settings
fullname	*varchar(30)*	Full name of the user
srvname	*varchar*(30)	Name of server to which a passthrough connection must be established if the AUTOCONNECT flag is turned on.

On the Adaptive Server distribution media, *syslogins* contains an entry in which the name is “sa”, the *suid* is 1, and the password is null. It also contains the entry “probe” with an unpublished password. The login “probe” and the user “probe” exist for the two phase commit probe process, which uses a challenge and response mechanism to access Adaptive Server.

Table 8-12 lists the bit representations for the *status* column:

Table 8-12: status control bits in the syslogins table

Decimal	Hex	Status
1	0x1	Password contains fewer than 6 characters or is NULL
2	0x2	Account is locked
4	0x4	Password has expired

Indexes

Unique clustered index on *suid*
Unique nonclustered index on *name*

Referenced by System Procedures

sp_addalias, sp_addlogin, sp_addremotelogin, sp_adduser, sp_audit, sp_changedbowner, sp_checknames, sp_checkreswords, sp_clearstats, sp_displaylogin, sp_droplogin, sp_helpdb, sp_helpuser, sp_locklogin, sp_modifylogin, sp_reportstats, sp_role

syslogs

(all databases)

Description

syslogs contains the transaction log. It is used by Adaptive Server for recovery and roll forward. It is not useful to users.

You cannot delete from, insert into, or update *syslogs*. Every data modification operation is logged, so before you can change *syslogs*, the change must be logged. This means that a change operation on *syslogs* adds a row to *syslogs*, which then must be logged, adding another row to *syslogs*, and so on, producing an infinite loop. The loop continues until the database becomes full.

Columns

Name	Datatype	Description
xactid	*binary(6)*	Transaction ID
op	*tinyint*	Number of update operation

Indexes

None

Referenced by System Procedures

None

syslogshold

(*master* database only)

Description

syslogshold contains information about each database's oldest active transaction (if any) and the Replication Server truncation point (if any) for the transaction log, but it is not a normal table. Rather, it is built dynamically when queried by a user. No updates to *syslogshold* are allowed.

Columns

Name	Datatype	Description
dbid	*smallint*	Database ID.
reserved	*int*	Unused.
spid	*smallint*	Server process ID of the user that owns the oldest active transaction (always 0 for Replication Server).
page	*int*	Starting page number of active portion in *syslogs* defined by oldest transaction (or the truncation page in *syslogs* for Replication Server).
xactid	*char(6)*	ID of the oldest active transaction (always 0x000000 for Replication Server).
masterxactid	*char(6)*	ID of the transaction's master transaction (if any) for multi-database transactions; otherwise 0x000000 (always 0x000000 for Replication Server).
starttime	*datetime*	Date and time the transaction started (or when the truncation point was set for Replication Server).

Name	Datatype	Description
name	*char(67)*	Name of the oldest active transaction. It is the name defined with **begin transaction**, "$user_transaction" if no value is specified with **begin transaction**, or "$chained_transaction" for implicit transactions started by the ANSI chained mode. Internal transactions started by Adaptive Server have names that begin with the dollar sign ($) and are named for the operation, or are named "$replication_truncation_point" for Replication Server.

Indexes

None

Referenced by System Procedures

None

sysmessages

(*master* database only)

Description

sysmessages contains one row for each system error or warning that can be returned by Adaptive Server. Adaptive Server displays the error description on the user’s screen.

Columns

Name	Datatype	Description
error	*int*	Unique error number
severity	*smallint*	Severity level of error
dlevel	*smallint*	Reserved
description	*varchar(255)*	Explanation of error with placeholders for parameters
langid	*smallint*	Language; null for us_english
sqlstate	*varchar(5)*	SQLSTATE value for the error

Indexes

Clustered index on *error, dlevel*
Unique nonclustered index on *error, dlevel, langid*

Referenced by System Procedures

sp_configure, sp_dboption, sp_depends, sp_droplanguage, sp_getmessage, sp_help, sp_helpdb, sp_helpdevice, sp_helpremotelogin, sp_remoteoption

sysmonitors

(*master* database only)

Description

sysmonitors contains one row for each monitor counter.

Columns

Name	Datatype	Description
field_name	*char(35)*	Name of the counter
group_name	*char(25)*	Group this counter belongs to
field_id	*smallint*	Unique identifier for the row
value	*int*	Current value of the counter
description	*char(255)*	Description of the counter; not used

Indexes

None

Referenced by System Procedures

sp_sysmon

sysobjects

(all databases)

Description

sysobjects contains one row for each table, view, stored procedure, extended stored procedure, log, rule, default, trigger, check constraint, referential constraint, and (in *tempdb* only) temporary object.

Columns

Name	Datatype	Description
name	*sysname*	Object name
id	*int*	Object ID
uid	*smallint*	User ID of object owner

type	*char(2)*	One of the following object types:
		D = default
		L = log
		P = procedure
		PR = prepare objects (created by Dynamic SQL)
		R = rule
		RI = referential constraint
		S = system table
		TR = trigger
		U = user table
		V = view
		XP = extended stored procedure

userstat	*smallint*	Application-dependent type information (32768 decimal [0x8000 hex] indicates to Data Workbench® that a procedure is a report)
sysstat	*smallint*	Internal status information (256 decimal [0x100 hex] indicates that table is read-only)
indexdel	*smallint*	Index delete count (incremented if an index is deleted)
schemacnt	*smallint*	Count of changes in the schema of an object (incremented if a rule or default is added)
sysstat2	*smallint*	Additional internal status information (see Table 8-13)
crdate	*datetime*	Date the object was created
expdate	*datetime*	Reserved

Name	Datatype	Description
deltrig	*int*	Stored procedure ID of a delete trigger if the entry is a table. Table ID if the entry is a trigger.
instrig	*int*	Stored procedure ID of a table's insert trigger if the entry is a table
updtrig	*int*	Stored procedure ID of a table's update trigger if the entry is a table
seltrig	*int*	Reserved
ckfirst	*int*	ID of first check constraint on the table
cache	*smallint*	Reserved
audflags	*int*	Object's audit settings
objspare	*int*	Spare
versionts	*binary*	

Table 8-13 lists the bit representations for the *sysstat2* column:

Table 8-13: sysstat2 control bits in the sysobjects table

Decimal	Hex	Status
1	0x1	Table has a referential constraint
2	0x2	Table has a foreign key constraint
4	0x4	Table has more than one check constraint
8	0x8	Table has a primary key constraint
16	0x10	Stored procedure can execute only in chained transaction mode
32	0x20	Stored procedure can execute in any transaction mode
64	0x40	Table has an IDENTITY field
512	0x200	Table does not contain variable-length columns

Indexes

Unique clustered index on *id*
Unique nonclustered index on *name, uid*

Referenced by System Procedures

sp_addmessage, sp_addthreshold, sp_audit, sp_bindefault, sp_bindmsg, sp_bindrule, sp_checknames, sp_checkreswords, sp_column_privileges, sp_columns, sp_commonkey, sp_depends, sp_dropgroup, sp_dropkey, sp_dropsegment, sp_dropthreshold, sp_droptype, sp_dropuser, sp_estspace, sp_fkeys, sp_foreignkey, sp_help, sp_helpconstraint, sp_helpindex, sp_helpjoins, sp_helpkey, sp_helprotect, sp_helpthreshold, sp_indsuspect, sp_modifythreshold,

sp_pkeys, sp_placeobject, sp_primarykey, sp_procxmode, sp_recompile, sp_relimit, sp_remap, sp_rename, sp_spaceused, sp_sproc_columns, sp_statistics, sp_stored_procedures, sp_table_privileges, sp_tables, sp_unbindefault, sp_unbindmsg, sp_unbindrule

syspartitions

(all databases)

Description

syspartitions contains one row for each partition (page chain) of a partitioned table.

Columns

Name	Datatype	Description
state	*smallint*	Internal information about the state of the partition
id	*int*	Object ID of the partitioned table
partitionid	*int*	Partition ID number
firstpage	*int*	Page number of the partition's first page
controlpage	*int*	Page number of the partition's control page
spare	*binary(32)*	Reserved

Indexes

Unique clustered index on *id*, *partitionid*

Referenced by System Procedures

sp_help, **sp_helppartition**, **sp_placeobject**

sysprocedures

(all databases)

Description

sysprocedures contains entries for each view, default, rule, trigger, procedure, declarative default, and check constraint. The plan or sequence tree for each object is stored in binary form. If the sequence tree does not fit into one entry, it is broken into more than one row. The *sequence* column identifies the sub-rows.

Columns

Name	Datatype	Description
type	*smallint*	Object type (see Table 8-14)
id	*int*	Object ID
sequence	*smallint*	Sequence number if more than one row is used to describe this object
status	*smallint*	Internal system status
number	*smallint*	Sub-procedure number when the procedure is grouped (0 for non-procedure entries)
version	*int*	

Table 8-14 lists the bit representations for the *type* column.

Table 8-14: type control bits in the sysprocedures table

Decimal	Hex	Status
1	0x1	Entry describes a plan (reserved)
2	0x2	Entry describes a tree

Indexes

Unique clustered index on *id, type, sequence, number*

Referenced by System Procedures

sp_bindefault, **sp_bindrule**, **sp_remap**, **sp_sproc_columns**, **sp_stored_procedures**, **sp_unbindefault**, **sp_unbindrule**

sysprocesses

(*master* database only)

Description

sysprocesses contains information about Adaptive Server processes, but it is not a normal table. Rather, it is built dynamically when queried by a user. No updates to *sysprocesses* are allowed.

Use the **kill** statement to kill a process.

Columns

Name	Datatype	Description
spid	*smallint*	Process ID
kpid	*int*	Kernel process ID
enginenum	*int*	Number of engine on which process is being executed
status	*char(12)*	Process ID status (see Table 8-15)
suid	*smallint*	Server user ID of user who issued command
hostname	*char(10)*	Name of host computer
program_name	*char(16)*	Name of front-end module
hostprocess	*char(8)*	Host process ID number
cmd	*char(16)*	Command currently being executed
cpu	*int*	Cumulative CPU time for process in ticks
physical_io	*int*	Number of disk reads and writes for current command
memusage	*int*	Amount of memory allocated to process
blocked	*smallint*	Process ID of blocking process, if any
dbid	*smallint*	Database ID
uid	*smallint*	ID of user who executed command
gid	*smallint*	Group ID of user who executed command
tran_name	*varchar(64)*	Name of the active transaction
time_blocked	*int*	Time blocked in seconds
network_pktsz	*int*	Current connection’s network packet size
fid	*smallint*	Process ID of the worker process’ parent
execlass	*varchar(30)*	Execution class that the process is bound to

Name	Datatype	Description
priority	*varchar(10)*	Base priority associated with the process
affinity	*varchar(30)*	Name of the engine to which the process has affinity
id	*int*	Object ID of the currently running procedure (or 0 if no procedure is running)
stmtnum	*int*	The current statement number within the running procedure (or the SQL batch statement number if no procedure is running)
linenum	*int*	The line number of the current statement within the running stored procedure (or the line number of the current SQL batch statement if no procedure is running)
origsuid	*smallint*	Original server user ID. If this value is not NULL, a user with an *suid* of *origsuid* executed **set proxy** or **set session authorization** to impersonate the user who executed the command.

Table 8-15 lists the values for the *status* column:

Table 8-15: sysprocesses status column values

Status	Meaning
alarm sleep	Waiting for alarm to wake process up (user executed a **waitfor delay** command)
background	A process, such as a threshold procedure, run by Adaptive Server rather than by a user process
infected	Server has detected a serious error condition; extremely rare
lock sleep	Waiting on a lock acquisition
log suspend	Processes suspended by reaching the last-chance threshold on the log
recv sleep	Waiting on a network read
runnable	In the queue of runnable processes
running	Actively running on one of the server engines
send sleep	Waiting on a network send
sleeping	Waiting on a disk I/O, or some other resource (often indicates a process that is running, but doing extensive disk I/O)
stopped	Stopped process
sync sleep	Waiting on a synchronization message from another process in the family

Indexes

None

Referenced by System Procedures

sp_dboption, **sp_droplogin**, **sp_locklogin**, **sp_role**, **sp_showplan**, **sp_who**

sysprotects

(all databases)

Description

sysprotects contains information on permissions that have been granted to, or revoked from, users, groups, and roles.

Columns

Name	Datatype	Description
id	*int*	ID of the object to which this permission applies.
uid	*smallint*	ID of the user, group, or role to which this permission applies.
action	*tinyint*	One of the following permissions: 167 = **set proxy** or **set session authorization** 193 = **select** 195 = **insert** 196 = **delete** 197 = **update** 224 = **execute** 151 = **references** 203 = **create database** 233 = **create default** 222 = **create procedure** 236 = **create rule** 198 = **create table** 207 = **create view** 228 = **dump database** 235 = **dump transaction**
protecttype	*tinyint*	One of the following values: 0 = **grant with grant** 1 = **grant** 2 = **revoke**
columns	*varbinary(32)*	Bitmap of columns to which this **select** or **update** permission applies. Bit 0 indicates all columns; 1 means permission applies to that column; NULL means no information.
grantor	*smallint*	User ID of the grantor (or of object owner if grantor is a System Administrator).

Indexes

Unique clustered index on *id, action, grantor, uid, protecttype*

Referenced by System Procedures

sp_changegroup, **sp_dropgroup**, **sp_dropuser**, **sp_helprotect**, **sp_stored_procedures**, **sp_tables**

sysreferences

(all databases)

Description

sysreferences contains one row for each referential integrity constraint declared on a table or column.

Columns

Name	Datatype	Description
indexid	*smallint*	ID of the unique index on referenced columns
constrid	*int*	Object ID of the constraint from *sysobjects*
tableid	*int*	Object ID of the referencing table
reftabid	*int*	Object ID of the referenced table
keycnt	*tinyint*	Number of columns in the foreign key
status	*smallint*	Reserved
frgndbid	*smallint*	Reserved
frgndbname	*varchar(30)*	Name of the database that includes the referencing table (the table with the foreign key); NULL if the referencing table is in the current database
pmrydbid	*smallint*	Reserved
pmrydbname	*varchar(30)*	Name of the database that includes the referenced table (the table with the primary key); NULL if the referenced table is in the current database
spare2	*int*	Reserved
fokey1 . . .	*tinyint*	Column ID of the first referencing column
fokey16	*tinyint*	Column ID of the sixteenth referencing column
refkey1 . . .	*tinyint*	Column ID of the first referenced column
refkey16	*tinyint*	Column ID of the sixteenth referenced column

Indexes

Clustered index on *tableid, frgndbname*
Unique nonclustered index on *frgndbid, constrid*
Nonclustered index on *reftabid, indexid, pmrydbname*

Referenced by System Procedures

sp_fkeys, **sp_helpconstraint**

sysremotelogins

(*master* database only)

Description

s*ysremotelogins* contains one row for each remote user that is allowed to execute remote procedure calls on this Adaptive Server.

Columns

Name	Datatype	Description
remoteserverid	*smallint*	Identifies the remote server
remoteusername	*varchar(30)*	User's login name on remote server
suid	*smallint*	Local server user ID
status	*smallint*	Bitmap of options

Indexes

Unique clustered index on *remoteserverid, remoteusername*

Referenced by System Procedures

sp_addremotelogin, **sp_checknames**, **sp_checkreswords**, **sp_dropremotelogin**, **sp_dropserver**, **sp_helpremotelogin**, **sp_remoteoption**

sysresourcelimits

(*master* database only)

Description

sysresourceslimits contains a row for each resource limit defined by Adaptive Server. Resource limits specify the maximum amount of server resources that can be used by a Adaptive Server login or an application to execute a query, query batch, or transaction.

Columns

Name	Datatype	Description
name	*varchar(30) null*	Login name
appname	*varchar(30) null*	Application name
rangeid	*smallint*	*id* column from *systimeranges*
limitid	*smallint*	*id* column from *spt_limit_types*
limitvalue	*int*	Value of limit
enforced	*tinyint*	Subset of the *enforced* column from *spt_limit_types*: 1 = prior to execution 2 = during execution 3 = both
actiontotake	*tinyint*	Action to take on a violation: 1 = issue warning 2 = abort query batch 3 = abort transaction 4 = kill session
scope	*tinyint*	Scope of user limit (a bitmap indicating one or more of the following): 1 = query 2 = query batch 4 = transaction
spare	*tinyint*	Reserved

Indexes

Clustered index on *name*, *appname*

Referenced by System Procedures

sp_add_resource_limit, sp_drop_resource_limit, sp_help_resource_limit, sp_modify_resource_limit

sysroles

(all databases)

Description

sysroles maps server role IDs to local role IDs.

Columns

Name	Datatype	Description
id	*smallint*	Server role ID (*srid*)
lrid	*smallint*	Local role ID
type	*smallint*	Unused
status	*smallint*	Unused

When a database permission is granted to a role, if an entry for the role does not exist in *syssrvroles*, Adaptive Server adds an entry to *sysroles* map the local role ID (*lrid*) to the server-wide role ID (*srid)* in *syssrvroles*.

Indexes

Unique clustered index on *lrid*

Referenced by System Procedures

None

syssecmechs

(*master* database only)

Description

syssecmechs contains information about the security services supported by each security mechanism that is available to Adaptive Server, but it is not a normal table. Rather, it is built dynamically when queried by a user.

Columns

Name	Datatype	Description
sec_mech_name	*varchar(30)*	Name of the security mechanism; for example, “NT LANMANAGER”
available_service	*varchar(30)*	Name of the security service supported by the security mechanism; for example, “unified login”

Indexes

None

Referenced by System Procedures

None

syssegments

(all databases)

Description

syssegments contains one row for each segment (named collection of disk pieces). In a newly created database, the entries are: segment 0 (*system*) for system tables; segment 2 *(logsegment*) for the transaction log; and segment 1 (*default*) for other objects.

Columns

Name	Datatype	Description
segment	*smallint*	Segment number
name	*sysname*	Segment name
status	*int null*	Indicates which segment is the default segment

Indexes

None

Referenced by System Procedures

sp_addsegment, sp_addthreshold, sp_checknames, sp_checkreswords, sp_dropsegment, sp_dropthreshold, sp_dropuser, sp_extendsegment, sp_helpdb, sp_helpindex, sp_helpsegment, sp_helpthreshold, sp_modifythreshold, sp_placeobject

sysservers

(*master* database only)

Description

sysservers contains one row for each remote Adaptive Server, Backup Server™, or Open Server™ on which this Adaptive Server can execute remote procedure calls.

Columns

Name	Datatype	Description
srvid	*smallint*	ID number (for local use only) of the remote server
srvstatus	*smallint*	Bitmap of options (see Table 8-14)
srvname	*varchar(30)*	Server name
srvnetname	*varchar(32)*	Interfaces file name for the server
srvclass	*smallint*	Server category defined by the class parameter of **sp_addserver**. See Table 8-17.
srvsecmech	*varchar(30)*	Security mechanism

Table 8-16 lists the bit representations for the *srvstatus* column:

Table 8-16: status control bits in the sysservers table

Decimal	Hex	Status
0	0x0	Timeouts are enabled
1	0x1	Timeouts are disabled
2	0x2	Network password encryption is enabled
4	0x4	Remote server is read only
8	0x8	Use rpc security model A

Table 8-17 lists the server categories for the *srvclass* column:

Table 8-17: Server categories in the sysservers table

srvclass	Server category
0	Local server (this server)
1	Another Adaptive Server or Component Integration Services server
3	Server coded to the DirectCONNECT specification
4	Server accessible by Net-Gateway or MDI Database Gateway
5	Server coded to the Generic Access Module specification

Indexes

Unique clustered index on *srvid*
Unique nonclustered index on *srvname*

Referenced by System Procedures

sp_addremotelogin, **sp_addserver**, **sp_checknames**, **sp_checkreswords**, **sp_configure**, **sp_dropremotelogin**, **sp_dropserver**, **sp_helpremotelogin**, **sp_helpserver**, **sp_remoteoption**, **sp_serveroption**

syssrvroles

(*master* database only)

Description

syssrvroles contains a row for each system or user-defined role.

Columns

Name	Datatype	Description
srid	*smallint*	Server role ID
name	*varchar*(30)	Name of the role
password	*varinary*(30)	Password for the role (encrypted)

Indexes

Unique clustered index on *srid*

Referenced by System Procedures

sp_adduser, sp_changegroup, sp_displaylogin, sp_dropgroup, sp_helpgroup, sp_role

systhresholds

(all databases)

Description

systhresholds contains one row for each threshold defined for the database.

Columns

Name	Datatype	Description
segment	*smallint*	Segment number for which free space is being monitored.
free_space	*int*	Size of threshold, in 2K pages (4K for Stratus).
status	*smallint*	Bit 1 equals 1 for the logsegment's last-chance threshold, 0 for all other thresholds.
proc_name	*varchar(255)*	Name of the procedure that is executed when the number of unused pages on *segment* falls below *free_space*.
suid	*smallint*	The server user ID of the user who added the threshold or modified it most recently.
currauth	*varbinary(255)*	A bit mask that indicates which roles were active for *suid* at the time the threshold was added or most recently modified. When the threshold is crossed, *proc_name* executes with this set of roles, less any that have been deactivated since the threshold was added or last modified.

Indexes

Unique clustered index on *segment, free_space*

Referenced by System Procedures

sp_addthreshold, **sp_dropsegment**, **sp_dropthreshold**, **sp_dropuser**, **sp_helpthreshold**, **sp_modifythreshold**

systimeranges

(*master* database only)

Description

systimeranges stores named time ranges, which are used by Adaptive Server to control when a resource limit is active.

Columns

Name	Datatype	Description
name	*varchar(30)*	Unique name of the time range.
id	*smallint*	Unique identifier for the time range. 1 represents the “at all times” limit.
startday	*tinyint*	Day of week (1–7) for the beginning of the range. Monday = 1, Sunday = 7.
endday	*tinyint*	Day of week (1–7) for the end of the range. Monday = 1, Sunday = 7.
starttime	*varchar(10)*	Time of day for the beginning of the range.
endtime	*varchar(10)*	Time of day for the end of the range.

Indexes

Clustered index on *id*

Referenced by System Procedures

sp_add_resource_limit, sp_add_time_range, sp_drop_resource_limit, sp_drop_time_range, sp_help_resource_limit, sp_modify_resource_limit, sp_modify_time_range

systypes

(all databases)

Description

systypes contains one row for each system-supplied and user-defined datatype. Domains (defined by rules) and defaults are given, if they exist.

The rows that describe system-supplied datatypes cannot be altered.

Columns

Name	Datatype	Description
uid	*smallint*	User ID of datatype creator
usertype	*smallint*	User type ID
variable	*bit*	1 if datatype is variable length; 0 otherwise
allownulls	*bit*	Indicates whether nulls are allowed for this datatype
type	*tinyint*	Physical storage datatype
length	*tinyint*	Physical length of datatype
tdefault	*int*	ID of system procedure that generates default for this datatype
domain	*int*	ID of system procedure that contains integrity checks for this datatype
name	*sysname*	Datatype name
printfmt	*varchar(255)*	Reserved
prec	*tinyint*	Number of significant digits
scale	*tinyint*	Number of digits to the right of the decimal point
ident	*tinyint*	1 if column has the IDENTITY property, 0 if it does not
hierarchy	*tinyint*	Precedence of the datatype in mixed mode arithmetic

Table 8-18 lists each system-supplied datatype's *name*, *hierarchy*, *type* (not necessarily unique), and *usertype* (unique). The datatypes are ordered by *hierarchy*. In mixed-mode arithmetic, the datatype with the lowest *hierarchy* takes precedence:

Table 8-18: Datatype names, hierarchy, types, and usertypes

Name	*hierarchy*	*type*	*usertype*
floatn	1	109	14
float	2	62	8
datetimn	3	111	15
datetime	4	61	12
real	5	59	23
numericn	6	108	28
numeric	7	63	10
decimaln	8	106	27
decimal	9	55	26
moneyn	10	110	17
money	11	60	11
smallmoney	12	122	21
smalldatetime	13	58	22
intn	14	38	13
int	15	56	7
smallint	16	52	6
tinyint	17	48	5
bit	18	50	16
varchar	19	39	2
sysname	19	39	18
nvarchar	19	39	25
char	20	47	1
nchar	20	47	24
varbinary	21	37	4
timestamp	21	37	80
binary	22	45	3
text	23	35	19
image	24	34	20

Indexes

Unique clustered index on *name*
Unique nonclustered index on *usertype*

Referenced by System Procedures

sp_addtype, sp_bindefault, sp_bindrule, sp_checknames, sp_checkreswords, sp_columns, sp_datatype_info, sp_droptype, sp_dropuser, sp_help, sp_rename, sp_special_columns, sp_sproc_columns,sp_unbindefault, sp_unbindrule

sysusages

(*master* database only)

Description

sysusages contains one row for each **disk allocation piece** assigned to a database. Each database contains a specified number of database (logical) page numbers. Each disk piece includes the segments on the Adaptive Server distribution media, segments 0 and 1.

The **create database** command checks *sysdevices* and *sysusages* to find available disk allocation pieces. One or more contiguous disk allocation pieces are assigned to the database, and the mapping is recorded in *sysusages*.

Columns

Name	Datatype	Description
dbid	*smallint*	Database ID
segmap	*int*	Bitmap of possible segment assignments
lstart	*int*	First database (logical) page number
size	*int*	Number of contiguous database (logical) pages
vstart	*int*	Starting virtual page number
pad	*smallint*	Unused
unreservedpgs	*int*	Free space not part of an allocated extent

Indexes

Unique clustered index on *dbid, lstart*
Unique nonclustered index on *vstart*

Referenced by System Procedures

sp_addsegment, **sp_addthreshold**, **sp_databases**, **sp_dropdevice**, **sp_dropsegment**, **sp_extendsegment**, **sp_helpdb**, **sp_helplog**, **sp_helpsegment**, **sp_logdevice**, **sp_modifythreshold**, **sp_spaceused**

sysusermessages

(all databases)

Description

sysusermessages contains one row for each user-defined message that can be returned by Adaptive Server.

Columns

Name	Datatype	Description
error	*int*	Unique error number. Must be 20,000 or higher.
uid	*smallint*	Server user ID (**suser_id**) of the message creator.
description	*varchar(255)*	User-defined message with optional placeholders for parameters.
langid	*smallint*	Language ID for this message; null for us_english.
dlevel	*smallint*	Stores the with_log bit, which is used to call the appropriate routine to log a message.

Indexes

Clustered index on *error*
Unique nonclustered index on *error, langid*

Referenced by System Procedures

sp_addmessage, **sp_bindmsg**, **sp_dropmessage**, **sp_getmessage**, **sp_helpconstraint**

sysusers

(all databases)

Description

sysusers contains one row for each user allowed in the database, and one row for each group or role.

Columns

Name	Datatype	Description
suid	*smallint*	Server user ID, copied from *syslogins*.
uid	*smallint*	User ID, unique in this database, is used for granting and revoking permissions. User ID 1 is "dbo".
gid	*smallint*	Group ID to which this user belongs. If *uid* = *gid*, this entry defines a group. The group "public" has *suid* = -2; all other groups have *suid* = - *gid*.
name	*sysname*	User or group name, unique in this database.
environ	*varchar(255)*	Reserved.

On the Adaptive Server distribution media, *master..sysusers* contains some initial users: "dbo", whose *suid* is 1 and whose *uid* is 1; "guest", whose *suid* is -1 and whose *uid* is 2; and "public", whose *suid* is -2 and whose *uid* is 0. In addition, both system-defined and user-defined roles (**sa_role**, **sso_role**, *role_name*) is listed in *sysusers*.

The user "guest" provides a mechanism for giving users that are not explicitly listed in *sysusers* access to the database with a restricted set of permissions. The "guest" entry in *master* means that any user with an account on Adaptive Server (that is, with an entry in *syslogins*) can access *master*.

The user "public" refers to all users. The keyword **public** is used with the **grant** and **revoke** commands to signify that permission is being given to or taken away from all users.

Indexes

Unique clustered index on *suid*
Unique nonclustered index on *name*
Unique nonclustered index on *uid*

Referenced by System Procedures

sp_addalias, **sp_addgroup**, **sp_adduser**, **sp_changedbowner**, **sp_changegroup**, **sp_checknames**, **sp_checkreswords**, **sp_column_privileges**, **sp_depends**, **sp_dropgroup**, **sp_droptype**, **sp_dropuser**, **sp_helpgroup**, **sp_helprotect**, **sp_helpuser**, **sp_indsuspect**, **sp_stored_procedures**, **sp_table_privileges**, **sp_tables**

Appendixes

A Expressions, Identifiers, and Wildcard Characters

This appendix describes Transact-SQL expressions, valid identifiers, and wildcard characters.

Expressions

An expression is a combination of one or more constants, literals, functions, column identifiers and / or variables, separated by operators, that returns a single value. Expressions can be of several types, including **arithmetic**, **relational**, **logical** (or **Boolean**), and **character string**. In some Transact-SQL clauses, a subquery can be used in an expression. A **case** expression can be used in an expression.

Table A-1 lists the types of expressions that are used in Adaptive Server syntax statements.

Table A-1: Types of expressions used in syntax statements

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

Arithmetic and Character Expressions

The general pattern for arithmetic and character expressions is:

```
{constant | column_name | function | (subquery)
   | (case_expression)}
       [{arithmetic_operator | bitwise_operator |
        string_operator | comparison_operator }
   {constant | column_name | function | (subquery)
   | case_expression}]...
```

Relational and Logical Expressions

A logical expression or relational expression returns TRUE, FALSE, or UNKNOWN. The general patterns are:

```
expression comparison_operator [any | all] expression
```

```
expression [not] in expression
```

```
[not]exists expression
```

```
expression [not] between expression and expression
```

```
expression [not] like "match_string"
   [escape "escape_character"]
```

```
not expression like "match_string"
   [escape "escape_character"]
```

```
expression is [not] null
```

```
not logical_expression
```

```
logical_expression {and | or} logical_expression
```

Operator Precedence

Operators have the following precedence levels, where 1 is the highest level and 6 is the lowest:

1. unary (single argument) - + ~
2. * / %
3. binary (two argument) + - & | ^
4. **not**
5. **and**
6. **or**

When all operators in an expression are at the same level, the order of execution is left to right. You can change the order of execution with parentheses—the most deeply nested expression is processed first.

Arithmetic Operators

Adaptive Server uses the following arithmetic operators:

Table A-2: Arithmetic operators

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulo (Transact-SQL extension)

Addition, subtraction, division, and multiplication can be used on exact numeric, approximate numeric, and money type columns.

The modulo operator cannot be used on *smallmoney, money, float* or *real* columns. Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

When you perform arithmetic operations on mixed datatypes, for example *float* and *int*, Adaptive Server follows specific rules for determining the type of the result. See Chapter 7, "System and User-Defined Datatypes," for more information.

Bitwise Operators

The bitwise operators are a Transact-SQL extension for use with integer type data. These operators convert each integer operand into its binary representation, then evaluate the operands column by column. A value of 1 corresponds to true; a value of 0 corresponds to false.

Table A-3 summarizes the results for operands of 0 and 1. If either operand is NULL, the bitwise operator returns NULL:

Table A-3: Truth tables for bitwise operations

& (and)	1	0
1	1	0
0	0	0

Table A-3: Truth tables for bitwise operations (continued)

| (or)	1	0
1	1	1
0	1	0

^ (exclusive or)	1	0
1	0	1
0	1	0

~ (not)	
1	FALSE
0	0

The examples in Table A-4 use two *tinyint* arguments, A = 170 (10101010 in binary form) and B = 75 (01001011 in binary form).

Table A-4: Examples of bitwise operations

Operation	Binary Form	Result	Explanation
(A & B)	10101010 01001011 ------------ 00001010	10	Result column equals 1 if both A and B are 1. Otherwise, result column equals 0.
(A | B)	10101010 01001011 ------------ 11101011	235	Result column equals 1 if either A or B, or both, is 1. Otherwise, result column equals 0
(A ^ B)	10101010 01001011 ------------ 11100001	225	Result column equals 1 if either A or B, but not both, is 1
(~A)	10101010 ------------ 01010101	85	All 1's are changed to 0's and all 0's to 1's

The String Concatenation Operator

The string operator **+** can be used to concatenate two or more character or binary expressions. For example:

1.
```
select Name = (au_lname + ", " + au_fname)
from authors
```

   Displays author names under the column heading *Name* in last-name first-name order, with a comma after the last name; for example, “Bennett, Abraham.”

2.
```
select "abc" + "" + "def"
```

   Returns the string “abc def”. The empty string is interpreted as a single space in all *char*, *varchar*, *nchar*, *nvarchar*, and *text* concatenation, and in *varchar* insert and assignment statements.

When concatenating non-character, non-binary expressions, always use **convert**:

```
select "The date is " +
    convert(varchar(12), getdate())
```

A string concatenated with NULL evaluates to the value of the string. This is an exception to the SQL standard, which states that a string concatenated with a NULL should evaluate to NULL.

The Comparison Operators

Adaptive Server uses the comparison operators listed in Table A-5:

Table A-5: Comparison operators

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
!=	Not equal to (Transact-SQL extension)
!>	Not greater than (Transact-SQL extension)
!<	Not less than (Transact-SQL extension)

In comparing character data, **<** means closer to the beginning of the server’s sort order and **>** means closer to the end of the sort order. Uppercase and lowercase letters are equal in a case-insensitive sort order. Use **sp_helpsort** to see the sort order for your Adaptive Server.

Trailing blanks are ignored for comparison purposes. So, for example, "Dirk" is the same as "Dirk ".

In comparing dates, **<** means earlier and **>** means later.

Put single or double quotes around all character and *datetime* data used with a comparison operator:

```
= "Bennet"
> "May 22 1947"
```

Nonstandard Operators

The following operators are Transact-SQL extensions:

- Modulo operator: **%**
- Negative comparison operators: **!>**, **!<**, **!=**
- Bitwise operators: **~**, **^**, **|**, **&**
- Join operators: ***=** and **=***

Using *any*, *all* and *in*

any is used with **<**, **>**, or **=** and a subquery. It returns results when any value retrieved in the subquery matches the value in the **where** or **having** clause of the outer statement. See Chapter 5, "Subqueries: Using Queries Within Other Queries," in the *Transact-SQL User's Guide* for more information.

all is used with **<** or **>** and a subquery. It returns results when all values retrieved in the subquery are less than (**<**) or greater than (**>**) the value in the **where** or **having** clause of the outer statement. See Chapter 5, "Subqueries: Using Queries Within Other Queries," in the *Transact-SQL User's Guide* for more information.

in returns results when any value returned by the second expression matches the value in the first expression. The second expression must be a subquery or a list of values enclosed in parentheses. **in** is equivalent to **= any**. See "where Clause" for details.

Negating and Testing

not negates the meaning of a keyword or logical expression.

Use **exists**, followed by a subquery, to test for the existence of a particular result.

Ranges

between is the range-start keyword; **and** is the range-end keyword. The range:

```
where column1 between x and y
```

is inclusive.

The range:

```
where column1 > x and column1 < y
```

is not inclusive.

Using Nulls in Expressions

Use **is null** or **is not null** in queries on columns defined to allow null values.

An expression with a bitwise or arithmetic operator evaluates to NULL if any of the operands are null. For example:

```
1 + column1
```

evaluates to NULL if *column1* is NULL.

Comparisons That Return TRUE

In general, the result of comparing null values is UNKNOWN, since it is not possible to determine whether NULL is equal (or not equal) to a given value or to another NULL. However, the following cases return TRUE when *expression* is any column, variable or literal, or combination of these, which evaluates as NULL:

- *expression* **is null**
- *expression* = **null**
- *expression* = *@x,* where *@x* is a variable or parameter containing NULL. This exception facilitates writing stored procedures with null default parameters.
- *expression* != *n,* where *n* is a literal that does not contain NULL, and *expression* evaluates to NULL.

The negative versions of these expressions return TRUE when the expression does not evaluate to NULL:

- *expression* **is not null**
- *expression* != **null**

- *expression* != *@x*

Note that the far right side of these exceptions is a literal null, or a variable or parameter containing NULL. If the far right side of the comparison is an expression (such as *@nullvar* + 1), the entire expression evaluates to NULL.

Following these rules, null column values do not join with other null column values. Comparing null column values to other null column values in a **where** clause always returns UNKNOWN for null values, regardless of the comparison operator, and the rows are not included in the results. For example, this query returns no result rows where column1 contains NULL in both tables (although it may return other rows):

```
select column1
from table1, table2
where table1.column1 = table2.column1
```

Difference Between FALSE and UNKNOWN

Although neither FALSE nor UNKNOWN returns values, there is an important logical difference between FALSE and UNKNOWN, because the opposite of false ("not false") is true. For example, "1 = 2" evaluates to false and its opposite, "1 != 2", evaluates to true. But "not unknown" is still unknown. If null values are included in a comparison, you cannot negate the expression to get the opposite set of rows or the opposite truth value.

Using "NULL" As a Character String

Only columns for which NULL was specified in the **create table** statement and into which you have explicitly entered NULL (no quotes), or into which no data has been entered, contain null values. Avoid entering the character string "NULL" (with quotes) as data for a character column. It can only lead to confusion. Use "N / A", "none", or a similar value instead. When you want to enter the value NULL explicitly, do **not** use single or double quotes.

NULLs Compared to the Empty String

The empty string (“ ”or ‘ ’) is always stored as a single space in variables and column data. This concatenation statement:

```
"abc" + "" + "def"
```

is equivalent to “abc def”, not to “abcdef”. The empty string is never evaluated as NULL.

Connecting Expressions

and connects two expressions and returns results when both are true. **or** connects two or more conditions and returns results when either of the conditions is true.

When more than one logical operator is used in a statement, **and** is evaluated before **or**. You can change the order of execution with parentheses.

Table A-6 shows the results of logical operations, including those that involve null values:

Table A-6: Truth tables for logical expressions

and	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
NULL	UNKNOWN	FALSE	UNKNOWN

or	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
NULL	TRUE	UNKNOWN	UNKNOWN

not	
TRUE	FALSE
FALSE	TRUE
NULL	UNKNOWN

The result UNKNOWN indicates that one or more of the expressions evaluates to NULL, and that the result of the operation cannot be determined to be either TRUE or FALSE. See "Using Nulls in Expressions" on page A-7 for more information.

Using Parentheses in Expressions

Parentheses can be used to group the elements in an expression. When "expression" is given as a variable in a syntax statement, a simple expression is assumed. "Logical expression" is specified when only a logical expression is acceptable.

Comparing Character Expressions

Character constant expressions are treated as *varchar*. If they are compared with non-*varchar* variables or column data, the datatype precedence rules are used in the comparison (that is, the datatype with lower precedence is converted to the datatype with higher precedence). If implicit datatype conversion is not supported, you must use the **convert** function.

Comparison of a *char* expression to a *varchar* expression follows the datatype precedence rule; the "lower" datatype is converted to the "higher" datatype. All *varchar* expressions are converted to *char* (that is, trailing blanks are appended) for the comparison.

Using the Empty String

The empty string ("") or ('') is interpreted as a single blank in **insert** or assignment statements on *varchar* data. In concatenation of *varchar, char, nchar, nvarchar* data, the empty string is interpreted as a single space; for example:

```
"abc" + "" + "def"
```

is stored as "abc def". The empty string is never evaluated as NULL.

Including Quotation Marks in Character Expressions

There are two ways to specify literal quotes within a *char* or *varchar* entry. The first method is to double the quotes. For example, if you begin a character entry with a single quote and you want to include a single quote as part of the entry, use two single quotes:

```
'I don''t understand.'
```

With double quotes:

```
"He said, ""It’s not really confusing."""
```

The second method is to enclose a quote in the opposite kind of quote mark. In other words, surround an entry containing a double quote with single quotes (or vice versa). Here are some examples:

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

Using the Continuation Character

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

Identifiers

Identifiers are names for database objects such as databases, tables, views, columns, indexes, triggers, procedures, defaults, rules, and cursors.

Adaptive Server identifiers can be a maximum of 30 bytes in length, whether single-byte or multibyte characters are used. The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_) character.

➤ ***Note***

Temporary table names, which begin with the pound sign (#), and local variable names, which begin with the at sign(@), are exceptions to this rule.

Subsequent characters can include letters, numbers, the symbols #, @, _, and currency symbols such as $ (dollars), ¥ (yen), and £ (pound sterling). Identifiers cannot include special characters such as !, %, ^, &, *, and . or embedded spaces.

You cannot use a reserved word, such as a Transact-SQL command, as an identifier. For a complete list of reserved words, see Appendix B, “Reserved Words.”

Tables Beginning with # (Temporary Tables)

Tables whose names begin with the pound sign (#) are temporary tables. You cannot create other types of objects whose names begin with the pound sign.

Adaptive Server performs special operations on temporary table names to maintain unique naming on a per-session basis. Long temporary table names are truncated to 13 characters (including the pound sign); short names are padded to 13 characters with underscores (_). A 17-digit numeric suffix that is unique for an Adaptive Server session is appended.

Case Sensitivity and Identifiers

Sensitivity to the case (upper or lower) of identifiers and data depends on the sort order installed on your Adaptive Server. Case sensitivity can be changed for single-byte character sets by reconfiguring Adaptive Server's sort order (see the *System Administration Guide* for more information). Case is significant in utility program options.

If Adaptive Server is installed with a case-insensitive sort order, you cannot create a table named *MYTABLE* if a table named MyTable or mytable already exists. Similarly, this command:

```
select * from MYTABLE
```

will return rows from *MYTABLE*, MyTable, or mytable, or any combination of uppercase and lowercase letters in the name.

Uniqueness of Object Names

Object names need not be unique in a database. However, column names and index names must be unique within a table, and other object names must be unique for each **owner** within a **database**. Database names must be unique on Adaptive Server.

Using Delimited Identifiers

Delimited identifiers are object names enclosed in double quotes. Using delimited identifiers allows you to avoid certain restrictions on object names. Table, view, and column names can be delimited by quotes; other object names cannot.

Delimited identifiers can be reserved words, can begin with non-alphabetic characters, and can include characters that would not otherwise be allowed. They cannot exceed 28 bytes.

◆ ***WARNING!***

Delimited identifiers may not be recognized by all front-end applications and should not be used as parameters to system procedures.

Before creating or referencing a delimited identifier, you must execute:

```
set quoted_identifier on
```

Each time you use the delimited identifier in a statement, you must enclose it in double quotes. For example:

```
create table "1one"(col1 char(3))
create table "include spaces" (col1 int)

create table "grant"("add" int)
insert "grant"("add") values (3)
```

While the **quoted_identifier** option is turned on, do not use double quotes around character or date strings; use single quotes instead. Delimiting these strings with double quotes causes Adaptive Server to treat them as identifiers. For example, to insert a character string into *col1* of *1table*, use:

```
insert "1one"(col1) values ('abc')
```

not:

```
insert "1one"(col1) values ("abc")
```

To insert a single quote into a column, use two consecutive single quotation marks. For example, to insert the characters “a’b” into *col1* use:

```
insert "1one"(col1) values('a''b')
```

Using Qualified Object Names

You can uniquely identify a table or column by adding other names that qualify it—the database name, owner’s name, and (for a column) the table or view name. Each qualifier is separated from the next one by a period. For example:

```
database.owner.table_name.column_name
database.owner.view_name.column_name
```

The naming conventions are:

```
[[database.]owner.]table_name
[[database.]owner.]view_name
```

Using Delimited Identifiers Within an Object Name

If you use **set quoted_identifier on**, you can use double quotes around individual parts of a qualified object name. Use a separate pair of quotes for each qualifier that requires quotes. For example, use:

```
database.owner."table_name"."column_name"
```

rather than:

```
database.owner."table_name.column_name"
```

Omitting the Owner Name

You can omit the intermediate elements in a name and use dots to indicate their positions, as long as the system is given enough information to identify the object:

```
database..table_name
database..view_name
```

Referencing Your Own Objects in the Current Database

You need not use the database name or owner name to reference your own objects in the current database. The default value for *owner* is the current user, and the default value for *database* is the current database.

If you reference an object without qualifying it with the database name and owner name, Adaptive Server tries to find the object in the current database among the objects you own.

Referencing Objects Owned by the Database Owner

If you omit the owner name and you do not own an object by that name, Adaptive Server looks for objects of that name owned by the Database Owner. You must qualify objects owned by the Database Owner only if you own an object of the same name, but you want to use the object owned by the Database Owner. However, you must qualify objects owned by other users with the user’s name, whether or not you own objects of the same name.

Using Qualified Identifiers Consistently

When qualifying a column name and table name in the same statement, be sure to use the same qualifying expressions for each; they are evaluated as strings and must match; otherwise, an error is returned. The second of the following examples is incorrect because the syntax style for the column name does not match the syntax style used for the table name.

```
1. select demo.mary.publishers.city
   from demo.mary.publishers

   city
   -----------------------
   Boston
   Washington
   Berkeley
```

```
2. select demo.mary.publishers.city
   from demo..publishers

   The column prefix "demo.mary.publishers" does not
   match a table name or alias name used in the query.
```

Determining Whether an Identifier Is Valid

Use the system function **valid_name**, after changing character sets or before creating a table or view, to determine whether the object name is acceptable to Adaptive Server. Here is the syntax:

```
select valid_name("Object_name")
```

If *object_name* is not a valid identifier (for example, if it contains illegal characters or is more than 30 bytes long), Adaptive Server returns 0. If *object_name* is a valid identifier, Adaptive Server returns a nonzero number.

Renaming Database Objects

Rename user objects (including user-defined datatypes) with **sp_rename**.

◆ ***WARNING!***

After you rename a table or column, be sure to redefine any procedures, triggers, and views that depend on the renamed object.

Using Multibyte Character Sets

In multibyte character sets, a wider range of characters is available for use in identifiers. For example, on a server with the Japanese language installed, the following types of characters may be used as the first character of an identifier: Zenkaku or Hankaku Katakana, Hiragana, Kanji, Romaji, Greek, Cyrillic, or ASCII.

Although Hankaku Katakana characters are legal in identifiers on Japanese systems, they are not recommended for use in heterogeneous systems. These characters cannot be converted between the EUC-JIS and Shift-JIS character sets.

The same is true for some 8-bit European characters. For example, the character "Œ," the OE ligature, is part of the Macintosh character set (codepoint 0xCE). This character does not exist in the ISO 8859-1 (iso_1) character set. If "Œ" exists in data being converted from the Macintosh to the ISO 8859-1 character set, it causes a conversion error.

If an object identifier contains a character that cannot be converted, the client loses direct access to that object.

Pattern Matching with Wildcard Characters

Wildcard characters represent one or more characters, or a range of characters, in a *match_string*. A *match_string* is a character string containing the pattern to find in the expression. It can be any combination of constants, variables, and column names or a concatenated expression, such as:

```
like @variable + "%".
```

If the match string is a constant, it must always be enclosed in single or double quotes.

Use wildcard characters with the keyword **like** to find character and date strings that match a particular pattern. You cannot use **like** to search for seconds or milliseconds (see "Using Wildcard Characters with datetime Data" on page A-22).

Use wildcard characters in **where** and **having** clauses to find character or date / time information that is **like**—or **not like**—the match string:

```
{where | having} [not]
   expression [not] like match_string
       [escape "escape_character"]
```

expression can be any combination of column names, constants, or functions with a character value.

Wildcard characters used without **like** have no special meaning. For example, this query finds any phone numbers that start with the four characters “415%”:

```
   select phone
   from authors
   where phone = "415%"
```

Using *not like*

Use **not like** to find strings that do not match a particular pattern. These two queries are equivalent: they find all the phone numbers in the *authors* table that do not begin with the 415 area code.

```
   select phone
   from authors
   where phone not like "415%"

   select phone
   from authors
   where not phone like "415%"
```

For example, this query finds the system tables in a database whose names begin with “sys”:

```
   select name
   from sysobjects
   where name like "sys%"
```

To see all the objects that are **not** system tables, use

```
    not like "sys%"
```

If you have a total of 32 objects and **like** finds 13 names that match the pattern, **not like** will find the 19 objects that do not match the pattern.

not like and the negative wildcard character [^] may give different results (see “The Caret (^) Wildcard Character” on page A-20). You cannot always duplicate **not like** patterns with **like** and **^**. This is because **not like** finds the items that do not match the entire **like**

pattern, but **like** with negative wildcard characters is evaluated one character at a time.

A pattern such as **like "[^s][^y][^s]%"** may not produce the same results. Instead of 19, you might get only 14, with all the names that begin with "s" **or** have "y" as the second letter **or** have "s" as the third letter eliminated from the results, as well as the system table names. This is because match strings with negative wildcard characters are evaluated in steps, one character at a time. If the match fails at any point in the evaluation, it is eliminated.

Case and Accent Insensitivity

If your Adaptive Server uses a case-insensitive sort order, case is ignored when comparing *expression* and *match_string*. For example, this clause:

```
where col_name like "Sm%"
```

would return "Smith," "smith," and "SMITH" on a case-insensitive Adaptive Server.

If your Adaptive Server is also accent-insensitive, it treats all accented characters as equal to each other and to their unaccented counterparts, both uppercase and lowercase. The **sp_helpsort** system procedure displays the characters that are treated as equivalent, displaying an "=" between them.

Using Wildcard Characters

You can use the match string with a number of wildcard characters, which are discussed in detail in the following sections. Table A-7 summarizes the wildcard characters:

Table A-7: Wildcard characters used with like

Symbol	Meaning
%	Any string of 0 or more characters
_	Any single character
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef])
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef])

Enclose the wildcard character and the match string in single or double quotes (**like** "[dD]eFr_nce").

The Percent Sign (%) Wildcard Character

Use the % wildcard character to represent any string of zero or more characters. For example, to find all the phone numbers in the *authors* table that begin with the 415 area code:

```
select phone
from authors
where phone like "415%"
```

To find names that have the characters “en” in them (Bennet, Green, McBadden):

```
select au_lname
from authors
where au_lname like "%en%"
```

Trailing blanks following “%” in a **like** clause are truncated to a single trailing blank. For example, “%” followed by two spaces matches “X ”(one space); “X ” (two spaces); “X   ” (three spaces), or any number of trailing spaces.

The Underscore (_) Wildcard Character

Use the _ wildcard character to represent any single character. For example, to find all six-letter names that end with “heryl” (for example, Cheryl):

```
select au_fname
from authors
where au_fname like "_heryl"
```

Bracketed ([ ]) Characters

Use brackets to enclose a range of characters, such as [a-f], or a set of characters such as [a2Br]. When ranges are used, all values in the sort order between (and including) *rangespec1* and *rangespec2* are returned. For example, “[0-z” matches 0-9, A-Z and a-z (and several punctuation characters) in 7-bit ASCII.

To find names ending with “inger” and beginning with any single character between M and Z:

```
select au_lname
from authors
where au_lname like "[M-Z]inger"
```

To find both “DeFrance” and “deFrance”:

```
select au_lname
from authors
where au_lname like "[dD]eFrance"
```

The Caret (^) Wildcard Character

The caret is the negative wildcard character. Use it to find strings that do not match a particular pattern. For example, “[^a-f]” finds strings that are not in the range a-f and “[^a2bR]” finds strings that are not “a,” “2,” “b,” or “R.”

To find names beginning with “M” where the second letter is not “c”:

```
select au_lname
from authors
where au_lname like "M[^c]%"
```

When ranges are used, all values in the sort order between (and including) *rangespec1* and *rangespec2* are returned. For example, “[0-z]” matches 0-9, A-Z , a-z, and several punctuation characters in 7-bit ASCII.

Using Multibyte Wildcard Characters

If the multibyte character set configured on your Adaptive Server defines equivalent double-byte characters for the wildcard characters _, %, - [,], and ^, you can substitute the equivalent character in the match string. The underscore equivalent represents either a single- or double-byte character in the match string.

Using Wildcard Characters As Literal Characters

To search for the occurrence of %, _, [,], or ^ within a string, you must use an escape character. When a wildcard character is used in conjunction with an escape character, Adaptive Server interprets the wildcard character literally, rather than using it to represent other characters.

Adaptive Server provides two types of escape characters:

- Square brackets (a Transact-SQL extension)
- Any single character that immediately follows an **escape** clause (compliant with the SQL standards)

Using Square Brackets As Escape Characters

Use square brackets as escape characters for the percent sign, the underscore, and the left bracket. The right bracket does not need an escape character; use it by itself. If you use the dash as a literal character, it must be the first character inside a set of square brackets.

Table A-8 shows some examples of square brackets as escape characters:

Table A-8: Using square brackets to search for wildcard characters

like **Predicate**	**Meaning**
like "5%"	5 followed by any string of 0 or more characters
like "5[%]"	5%
like "_n"	an, in, on (and so on)
like "[_]n"	_n
like "[a-cdf]"	a, b, c, d, or f
like "[-acdf]"	-, a, c, d, or f
like "[[]"	[
like "]"	]
like “[[]ab]”	[]ab

Using the *escape* Clause

Use the **escape** clause to specify an escape character. Any single character in the server’s default character set can be used as an escape character. If you try to use more than one character as an escape character, Adaptive Server generates an exception.

Do not use existing wildcard characters as escape characters because:

- If you specify the underscore (_) or percent sign (%) as an escape character, it loses its special meaning within that **like** predicate and acts only as an escape character.
- If you specify the left or right bracket ([or]) as an escape character, the Transact-SQL meaning of the bracket is disabled within that **like** predicate.
- If you specify the hyphen or caret (**-** or ^) as an escape character, it loses its special meaning and acts only as an escape character.

An escape character retains its special meaning within square brackets, unlike wildcard characters such as the underscore, the percent sign, and the open bracket.

The escape character is valid only within its **like** predicate and has no effect on other **like** predicates contained in the same statement. The only characters that are valid following an escape character are the wildcard characters (_, %, [,], or [^]), and the escape character itself. The escape character affects only the character following it, and subsequent characters are not affected by it.

If the pattern contains two literal occurrences of the character that happens to be the escape character, the string must contain four consecutive escape characters. If the escape character does not divide the pattern into pieces of one or two characters, Adaptive Server returns an error message.

Following are examples of **like** predicates with **escape** clauses:.

Table A-9: Using the escape clause

like Predicate	Meaning
like "5@%" escape "@"	5%
like "*_n" escape "*"	_n
like "%80@%%" escape "@"	String containing 80%
like "*_sql**%" escape "*"	String containing _sql*
like "%#####_#%%" escape "#"	String containing ##_%

To enforce standard behavior and disable the special meaning of the square brackets, use **set fipsflagger on**.

Using Wildcard Characters with *datetime* Data

When you use **like** with *datetime* values, Adaptive Server converts the dates to the standard *datetime* format, and then to *varchar*. Since the standard storage format does not include seconds or milliseconds, you cannot search for seconds or milliseconds with **like** and a pattern.

It is a good idea to use **like** when you search for *datetime* values, since *datetime* entries may contain a variety of date parts. For example, if you insert the value “9:20” and the current date into a column named *arrival_time*, the clause:

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

would not find the value, because Adaptive Server converts the entry into “Jan 1 1900 9:20AM.” However, the following clause would find this value:

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

B Reserved Words

Keywords, also known as reserved words, are words that have special meanings. Transact-SQL and SQL92 keywords are listed in this appendix.

Transact-SQL Keywords

The words in Table B-1 are reserved by Adaptive Server as keywords (part of SQL command syntax) and cannot be used as names of database objects such as databases, tables, rules, and defaults. They can be used as names of local variables and as stored procedure parameter names. You can use the system procedure **sp_checkreswords** to find the names of existing objects that are reserved words.

Table B-1: Transact-SQL keywords

activation	char_convert	dbcc	exclusive
add	check	deallocate	exec
all	checkpoint	declare	execute
alter	close	default	exists
and	clustered	delete	exit
any	commit	desc	external
arith_overflow	compute	disk	fetch
as	confirm	distinct	fillfactor
asc	connect	double	for
at	constraint	dummy	foreign
authorization	consumers	dump	from
avg	continue	else	goto
begin	controlrow	end	grant
between	convert	endtran	group
break	count	errlvl	having
browse	create	errordata	holdlock
bulk	current	errorexit	identity_insert
by	cursor	escape	identity_start
cascade	database	except	if

Table B-1: Transact-SQL keywords (continued)

in	off	read	temp
index	offsets	readtext	temporary
insert	on	reconfigure	textsize
intersect	once	references	to
into	online	replace	tran
is	only	return	transaction
isolation	open	revoke	trigger
key	option	role	truncate
kill	or	rollback	tsequal
level	order	rowcount	union
like	over	rows	unique
lineno	partition	rule	unpartition
load	passwd	save	update
max	perm	schema	use
max_rows_per_page	permanent	select	user
membership	plan	session	user_option
min	precision	set	using
mirror	prepare	setuser	values
mirrorexit	primary	shared	varying
national	print	shutdown	view
noholdlock	privileges	some	waitfor
nonclustered	proc	statistics	where
not	procedure	stripe	while
null	processexit	sum	with
lineno	proxy	syb_identity	work
numeric_transaction	public	syb_restree	writetext
of	raiserror	table	

SQL92 Keywords

Adaptive Server includes entry-level SQL92 features. Full SQL92 implementation includes the words listed in the following tables as command syntax. Since upgrading identifiers can be a complex process, we are providing this list for your convenience. The publication of this information does not commit Sybase to providing all of these SQL92 features in subsequent releases. In addition, subsequent releases may include keywords not included in this list.

The words in Table B-2 are SQL92 keywords that are not reserved words in Transact-SQL.

Table B-2: SQL92 keywords

absolute	corresponding	float
action	cross	found
allocate	current_date	full
are	current_time	get
assertion	current_timestamp	global
bit	current_user	go
bit_length	date	hour
both	day	immediate
cascaded	dec	indicator
case	decimal	initially
cast	deferrable	inner
catalog	deferred	input
char	describe	insensitive
char_length	descriptor	int
character	diagnostics	integer
character_length	disconnect	interval
coalesce	domain	join
collate	end-exec	language
collation	exception	last
column	extract	leading
connection	false	left
constraints	first	local

Table B-2: SQL92 keywords (continued)

lower	prior	timestamp
match	real	timezone_hour
minute	relative	timezone_minute
module	restrict	trailing
month	right	translate
names	scroll	translation
natural	second	trim
nchar	section	true
next	session_user	unknown
no	size	upper
nullif	smallint	usage
numeric	space	value
octet_length	sql	varchar
outer	sqlcode	when
output	sqlerror	whenever
overlaps	sqlstate	write
pad	substring	year
partial	system_user	zone
position	then	
preserve	time	

Potential SQL92 Reserved Words

If you are using the ISO / IEC 9075:1989 standard, also avoid using the words listed in Table B-3, as these words may become SQL92 reserved words in the future.

Table B-3: Potential SQL92 reserved words

after	loop	returns
alias	modify	routine
async	new	row
before	none	savepoint
boolean	object	search
breadth	oid	sensitive
call	old	sequence
completion	operation	signal
cycle	operators	similar
data	others	sqlexception
depth	parameters	structure
dictionary	pendant	test
each	preorder	there
elseif	private	type
equals	protected	under
general	recursive	variable
ignore	ref	virtual
leave	referencing	visible
less	resignal	wait
limit	return	without

C SQLSTATE Codes and Messages

This appendix describes Adaptive Server's SQLSTATE status codes and their associated messages. SQLSTATE codes are required for entry level SQL92 compliance. They provide diagnostic information about two types of conditions:

- **Warnings** – conditions that require user notification but are not serious enough to prevent a SQL statement from executing successfully
- **Exceptions** – conditions that prevent a SQL statement from having any effect on the database

Each SQLSTATE code consists of a 2-character class followed by a 3-character subclass. The class specifies general information about error type; the subclass specifies more specific information.

SQLSTATE codes are stored in the *sysmessages* system table, along with the messages that display when these conditions are detected. Not all Adaptive Server error conditions are associated with a SQLSTATE code—only those mandated by SQL92. In some cases, multiple Adaptive Server error conditions are associated with a single SQLSTATE value.

Warnings

Adaptive Server currently detects only one SQLSTATE warning condition, which is described in Table C-1:

Table C-1: SQLSTATE warnings

Message	Value	Description
Warning - null value eliminated in set function.	01003	Occurs when you use an aggregate function (**avg**, **max**, **min**, **sum**, or **count**) on an expression with a null value.

Exceptions

Adaptive Server detects the following types of exceptions:

- Cardinality violations
- Data exceptions

- Integrity constraint violations
- Invalid cursor states
- Syntax errors and access rule violations
- Transaction rollbacks
- **with check option** violations

Exception conditions are described in Table C-2 through Table C-8. Each class of exceptions appears in its own table. Within each table, conditions are sorted alphabetically by message text.

Cardinality Violations

Cardinality violations occur when a query that should return only a single row returns more than one row to an Embedded SQL™ application.

Table C-2: Cardinality violations

Message	Value	Description
Subquery returned more than 1 value. This is illegal when the subquery follows =, !=, <, <=, >, >=. or when the subquery is used as an expression.	21000	Occurs when: • A scalar subquery or a row subquery returns more than one row. • A **select into parameter_list** query in Embedded SQL returns more than one row.

Data Exceptions

Data exceptions occur when an entry:

- Is too long for its datatype,
- Contains an illegal escape sequence, or
- Contains other format errors.

Table C-3: Data exceptions

Message	Value	Description
Arithmetic overflow occurred.	22003	Occurs when: • An exact numeric type would lose precision or scale as a result of an arithmetic operation or **sum** function. • An approximate numeric type would lose precision or scale as a result of truncation, rounding, or a **sum** function.
Data exception - string data right truncated.	22001	Occurs when a *char* or *varchar* column is too short for the data being inserted or updated and non-blank characters must be truncated.
Divide by zero occurred.	22012	Occurs when a numeric expression is being evaluated and the value of the divisor is zero.
Illegal escape character found. There are fewer bytes than necessary to form a valid character.	22019	Occurs when you are searching for strings that match a given pattern if the escape sequence does not consist of a single character.
Invalid pattern string. The character following the escape character must be percent sign, underscore, left square bracket, right square bracket, or the escape character.	22025	Occurs when you are searching for strings that match a particular pattern when: • The escape character is not immediately followed by a percent sign, an underscore, or the escape character itself, or • The escape character partitions the pattern into substrings whose lengths are other than 1 or 2 characters.

Integrity Constraint Violations

Integrity constraint violations occur when an **insert**, **update**, or **delete** statement violates a **primary key**, **foreign key**, **check**, or **unique** constraint or a unique index.

Table C-4: Integrity constraint violations

Message	Value	Description
Attempt to insert duplicate key row in object *object_name* with unique index *index_name*	23000	Occurs when a duplicate row is inserted into a table that has a unique constraint or index.

Table C-4: Integrity constraint violations (continued)

Message	Value	Description
Check constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **update** or **delete** would violate a check constraint on a column.
Dependent foreign key constraint violation in a referential integrity constraint. dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **update** or **delete** on a primary key table would violate a foreign key constraint.
Foreign key constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	Occurs when an **insert** or **update** on a foreign key table is performed without a matching value in the primary key table.

Invalid Cursor States

Invalid cursor states occur when:

- A **fetch** uses a cursor that is not currently open, or
- An **update where current of** or **delete where current of** affects a cursor row that has been modified or deleted, or
- An **update where current of** or **delete where current of** affects a cursor row that not been fetched.

Table C-5: Invalid cursor states

Message	Value	Description
Attempt to use cursor *cursor_name* which is not open. Use the system stored procedure sp_cursorinfo for more information.	24000	Occurs when an attempt is made to fetch from a cursor that has never been opened or that was closed by a **commit** statement or an implicit or explicit **rollback**. Reopen the cursor and repeat the **fetch**.
Cursor *cursor_name* was closed implicitly because the current cursor position was deleted due to an update or a delete. The cursor scan position could not be recovered. This happens for cursors which reference more than one table.	24000	Occurs when the join column of a multitable cursor has been deleted or changed. Issue another **fetch** to reposition the cursor.

Table C-5: Invalid cursor states (continued)

Message	Value	Description
The cursor *cursor_name* had its current scan position deleted because of a DELETE / UPDATE WHERE CURRENT OF or a regular searched DELETE / UPDATE. You must do a new FETCH before doing an UPDATE or DELETE WHERE CURRENT OF.	24000	Occurs when a user issues an **update/delete where current of** whose current cursor position has been deleted or changed. Issue another **fetch** before retrying the **update/delete where current of**.
The UPDATE / DELETE WHERE CURRENT OF failed for the cursor *cursor_name* because it is not positioned on a row.	24000	Occurs when a user issues an **update/delete where current of** on a cursor that: • Has not yet fetched a row • Has fetched one or more rows after reaching the end of the result set

Syntax Errors and Access Rule Violations

Syntax errors are generated by SQL statements that contain unterminated comments, implicit datatype conversions not supported by Adaptive Server or other incorrect syntax.

Access rule violations are generated when a user tries to access an object that does not exist or one for which he or she does not have the correct permissions.

Table C-6: Syntax errors and access rule violations

Message	Value	Description
command permission denied on object *object_name*, database *database_name*, owner *owner_name*.	42000	Occurs when a user tries to access an object for which he or she does not have the proper permissions.
Implicit conversion from datatype '*datatype*' to '*datatype*' is not allowed. Use the CONVERT function to run this query.	42000	Occurs when the user attempts to convert one datatype to another but Adaptive Server cannot do the conversion implicitly.
Incorrect syntax near *object_name*.	42000	Occurs when incorrect SQL syntax is found near the object specified.
Insert error: column name or number of supplied values does not match table definition.	42000	Occurs during inserts when an invalid column name is used or when an incorrect number of values is inserted.
Missing end comment mark '*/'.	42000	Occurs when a comment that begins with the /* opening delimiter does not also have the */ closing delimiter.

Table C-6: Syntax errors and access rule violations (continued)

Message	Value	Description
object_name not found. Specify owner.objectname or use sp_help to check whether the object exists (sp_help may produce lots of output).	42000	Occurs when a user tries to reference an object that he or she does not own. When referencing an object owned by another user, be sure to qualify the object name with the name of its owner.
The size (*size*) given to the *object_name* exceeds the maximum. The largest size allowed is *size*.	42000	Occurs when: • The total size of all the columns in a table definition exceeds the maximum allowed row size. • The size of a single column or parameter exceeds the maximum allowed for its datatype.

Transaction Rollbacks

Transaction rollbacks occur when the **transaction isolation leve**l is set to 3, but Adaptive Server cannot guarantee that concurrent transactions can be serialized. This type of exception generally results from system problems such as disk crashes and offline disks.

Table C-7: Transaction rollbacks

Message	Value	Description
Your server command (process id #*process_id*) was deadlocked with another process and has been chosen as deadlock victim. Re-run your command.	40001	Occurs when Adaptive Server detects that it cannot guarantee that two or more concurrent transactions can be serialized.

with check option Violation

This class of exception occurs when data being inserted or updated through a view would not be visible through the view.

Table C-8: with check option violation

Message	Value	Description
The attempted insert or update failed because the target view was either created WITH CHECK OPTION or spans another view created WITH CHECK OPTION. At least one resultant row from the command would not qualify under the CHECK OPTION constraint.	44000	Occurs when a view, or any view on which it depends, was created with a **with check option** clause.

Index

Index

This index pertains to all three volumes of the *Adaptive Server Reference Manual*. It is divided into three sections:

- Symbols

  Indexes entries that begin with symbols.

- Numerics

  Indexes entries that begin numerically.

- Subjects

  Indexes subjects alphabetically.

Page numbers in **bold** are primary references.

Symbols