SYBASE®

Reference Manual
Volume 1: Building Blocks

Adaptive Server Enterprise

12.5

DOCUMENT ID: 36271-01-1250-01

LAST REVISED: June 2001

Contents

About This Book

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

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

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

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

Volume 4, "*System Tables*," provides reference information about the system tables, which store information about your server, databases, users, and other details of your server. It provides information about the tables in the dbccdb and dbccalt databases.

Audience

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

How to use this book

- Chapter 1, "System and User-Defined Datatypes," which describes the system and user-defined datatypes that are supplied with Adaptive Server and indicates how to use them to create user-defined datatypes.
- Chapter 2, "Transact-SQL Functions," lists the Adaptive Server functions in a table that provides the name and a brief description. Click on a function name in the table to go directly to the function.
- Chapter 3 through Chapter 6 provide manual pages for the individual functions.
- Chapter 7, "Expressions, Identifiers, and Wildcard Characters," which provides information about using the Transact-SQL language.

- Chapter 8, “Reserved Words,” which provides information about the Transact-SQL and SQL92 keywords.
- Chapter 9, “SQLSTATE Codes and Messages,” which contains information about Adaptive Server’s SQLSTATE status codes and the associated messages.

Related documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

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

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.
- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *What’s New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5, the system changes added to support those features, and the changes that may affect your existing applications.
- *Transact-SQL User’s Guide* – documents Transact-SQL, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.
- *Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.

- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- The *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book. Available only in print version.
- The *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using Sybase's DTM XA interface with X/Open XA transaction managers.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.

- *Sybase jConnect for JDBC Programmer's Reference* – describes the jConnect for JDBC product and explains how to use it to access data stored in relational database management systems.
- *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Historical Server User's Guide* –describes how to use Historical Server to obtain performance information for SQL Server and Adaptive Server.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.
- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.

Other sources of information

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

- Technical Library CD contains product manuals and is included with your software. The DynaText browser (downloadable from Product Manuals at http://www.sybase.com/detail/1,3693,1010661,00.html) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase/Powersoft newsgroups.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Conventions

The following sections describe conventions used in this manual.

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

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

Table 1: Font and syntax conventions for this manual

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

Variables, or words that stand for values that you fill in, are in italics.	`select` *`column_name`*
	`from` *`table_name`*
	`where` *`search_conditions`*

Type parentheses as part of the command.	`compute` *`row_aggregate`* `(`*`column_name`*`)`
Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates "is defined as".	::=
Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.	`[cash | check | credit]`
The comma means you may choose as many of the options shown as you want. Separate your choices with commas as part of the command.	`cash, check, credit`
The pipe or vertical bar(|) means you may select only one of the options shown.	`cash | check | credit`

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

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

  sp_dropdevice [*device_name*]

  or, for a command with more options:

  select *column_name*
     from *table_name*
     where *search_conditions*

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

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

```
select * from publishers
```

- Examples of output from the computer appear as follows:

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

(3 rows affected)
```

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

Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see the *System Administration Guide*.

If you need help

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

CHAPTER 1

System and User-Defined Datatypes

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

- Datatype categories
- Range and storage size
- Declaring the datatype of a column, variable, or parameter
- Datatype of mixed-mode expressions
- Converting one datatype to another
- Standards and compliance
- Exact numeric datatypes
- Approximate numeric datatypes
- Money datatypes
- Timestamp datatype
- Date and time datatypes
- Character datatypes
- Binary datatypes
- bit datatype
- sysname datatype
- text and image datatypes
- User-defined datatypes

Datatype categories

Adaptive Server provides several system datatypes and the user-defined datatypes timestamp and sysname. Table 1-1 lists the categories of Adaptive Server datatypes. Each category is described in a section of this chapter.

Table 1-1: Datatype categories

Category	Used for
Exact numeric datatypes	Numeric values (both integers and numbers with a decimal portion) that must be represented exactly
Approximate numeric datatypes	Numeric data that can tolerate rounding during arithmetic operations
Money datatypes	Monetary data
Timestamp datatype	Tables that are browsed in Client-Library™ applications
Date and time datatypes	Date and time information
Character datatypes	Strings consisting of letters, numbers, and symbols
Binary datatypes	Raw binary data, such as pictures, in a hexadecimal-like notation
bit 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 the maximum column size provided by you server's logical page size.
User-defined datatypes	Defining objects that inherit the rules, default, null type, IDENTITY property, and base datatype

Range and storage size

Table 1-2 lists the system-supplied datatypes and their synonyms and provides information about the range of valid values and storage size for each. For simplicity, the datatypes are printed in lowercase characters, although Adaptive Server allows you to use either uppercase or lowercase characters for system datatypes. User-defined datatypes, such as timestamp, are *case sensitive*. Most Adaptive Server-supplied datatypes are not reserved words and can be used to name other objects.

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

Datatypes	Synonyms	Range	Bytes of storage
Exact numeric datatypes			

Datatypes	Synonyms	Range	Bytes of storage
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	Determined by the maximum colum size for your server's logical page size	n
varchar(n)	char[acter] varying	Determined by the maximum colum size for your server's logical page size	actual entry length
unichar	Unicode character	Determined by the maximum colum size for your server's logical page size	n*@@unicharsize (@@unicharsize equals 2)
univarchar	unichar(acter) varying	Determined by the maximum colum size for your server's logical page size	actual number of characters *@@unicharsize
nchar(n)	national char[acter]	Determined by the maximum colum size for your server's logical page size	n * @@ncharsize
nvarchar(n)	nchar varying, national char[acter] varying	Determined by the maximum colum size for your server's logical page size	n
Binary datatypes			
binary(n)		Determined by the maximum colum size for your server's logical page size	n
varbinary(n)		Determined by the maximum colum size for your server's logical page size	actual entry length
Bit datatype			

Datatypes	**Synonyms**	**Range**	**Bytes of storage**
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 the logical page size
image		2^{31} -1 (2,147,483,647) bytes or fewer	0 until initialized, then a multiple of the logical page size

Declaring the datatype of a column, variable, or parameter

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

Declaring the datatype for a column in a table

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

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

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

For example:

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

Declaring the datatype for a local variable in a batch or procedure

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

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

For example:

```
declare @hope money
```

Declaring the datatype for a parameter in a stored procedure

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

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

For example:

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

Determining the datatype of a literal

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

- Literals between 2^{31} - 1 and -2^{31} with no decimal point are treated as integer.

- Literals that include a decimal point, or that fall outside the range for integers, are treated as numeric.

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

Datatype of mixed-mode expressions

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

Determining the datatype hierarchy

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

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

```
select name,hierarchy
from systypes
order by hierarchy
name                          hierarchy
----------------------------  ---------
floatn                                1
float                                 2
datetimn                              3
datetime                              4
real                                  5
numericn                              6
numeric                               7
decimaln                              8
decimal                               9
moneyn                               10
money                                11
smallmoney                           12
smalldatetime                        13
```

```
intn                                        14
int                                         15
smallint                                    16
tinyint                                     17
bit                                         18
univarchar                                  19
unichar                                     20
reserved                                    21
varchar                                     22
sysname                                     22
nvarchar                                    22
char                                        23
nchar                                       23
varbinary                                   24
timestamp                                   24
binary                                      25
text                                        26
image                                       27
(28 rows affected)
```

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

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

smallint(qty) * int(royalty) = int

Determining precision and scale

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

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

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

Table 1-3: Precision and scale after arithmetic operations

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

Converting one datatype to another

Many conversions from one datatype to another are handled automatically by Adaptive Server. These are called implicit conversions. Other conversions must be performed explicitly with the convert, inttohex, and hextoint functions. See “Datatype conversion functions” for details about datatype conversions supported by Adaptive Server.

Automatic conversion of fixed-length NULL columns

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

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

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

Original Fixed-Length Datatype	Converted To
char	varchar
unichar	univarchar
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	Complience 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*,*unichar*, *univarchar*, *sysname*, *text*, *timestamp*, and user-defined datatypes are Transact-SQL extensions.

Exact numeric datatypes

Function

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

Integer Types

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

Table 1-5: Integer datatypes

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

Entering integer data

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

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

Table 1-6: Valid integer values

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

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

Table 1-7: Invalid integer values

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

Decimal datatypes

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

Specifying precision and scale

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

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

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

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

- The scale specifies the maximum number of digits that can be stored to the right of the decimal point. The scale must be less than or equal to the precision. You can specify a scale ranging from 0 digits to 38 digits or use the default scale of 0 digits.

Storage size

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

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

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

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

Entering decimal data

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

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

Table 1-8: Valid decimal values

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

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

Table 1-9: Invalid decimal values

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

Standards and compliance

Standard	Complience 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”. To use these datatypes, you must understand their limitations.

When 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 are doing repeated calculations using approximate numeric datatypes—the results can be surprisingly and unexpectedly inaccurate.

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

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

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

Range, precision, and storage size

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

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

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

Table 1-10: Approximate numeric datatypes

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

Entering approximate numeric data

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

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

The value represented by the entry is the following product:

```
mantissa * 10EXPONENT
```

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

Values that may be entered by Open Client clients

"NaN" and "Inf" are special values that the floating point number standard uses to represent values that are "not a number" and "infinity," respectively. Adaptive Server does not usually permit these values, but Open Client clients can sometimes stuff these values into tables.

Standards and compliance

Standard	Complience level
SQL92	The float, double precision, and real datatypes are entry level compliant.

Money datatypes

Function

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

Accuracy

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

Range and storage size

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

Table 1-11: Money datatypes

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

Entering monetary values

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

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

Standards and compliance

Standard	Complience level
SQL92	The money and smallmoney datatypes are Transact-SQL extensions.

Timestamp datatype

Function

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

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)
```

Date and time datatypes

Function

Use datetime and smalldatetime to store absolute date and time information. Use timestamp to store binary-type information

Range and storage requirements

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

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

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

Entering *datetime* and *smalldatetime* data

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

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

Entering the date portion of a datetime or smalldatetime value

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

- You can enter the entire date as an unseparated string of 4, 6, or 8 digits, or use slash (/), hyphen (-), or period (.) separators between the date parts.
  - When entering dates as unseparated strings, use the appropriate format for that string length. Use leading zeros for single-digit years, months, and days. Dates entered in the wrong format may be misinterpreted or result in errors.
  - When entering dates with separators, use the set dateformat option to determine the expected order of date parts. If the first date part in a separated string is four digits, Adaptive Server interprets the string as *yyyy-mm-dd* format.
- Some date formats accept 2-digit years (*yy*):
  - Numbers less than 50 are interpreted as 20*yy*. For example, `01` is 2001, `32` is 2032, and `49` is 2049.
  - Numbers equal to or greater than 50 are interpreted as 19*yy*. For example, `50` is 1950, `74` is 1974, and `99` is 1999.
- You can specify the month as either a number or a name. Month names and their abbreviations are language-specific and can be entered in uppercase, lowercase, or mixed case.
- If you omit the date portion of a datetime or smalldatetime value, Adaptive Server uses the default date of January 1, 1900.

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

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

Date Format	Interpretation	Sample Entries	Meaning
4-digit string with no separators	Interpreted as *yyyy*. Date defaults to Jan 1 of the specified year.	"1947"	Jan 1 1947
6-digit string with no separators	Interpreted as *yymmdd*. For *yy* < 50, year is 20*yy*. For *yy* >= 50, year is 19*yy*.	"450128" "520128"	Jan 28 2045 Jan 28 1952
8-digit string with no separators	Interpreted as *yyyymmdd*.	"19940415"	Apr 15 1994
String consisting of 2-digit month, day, and year separated by slashes, hyphens, or periods, or a combination of the above.	The dateformat and language set options determine the expected order of date parts. For us_english, the default order is *mdy*. For *yy* < 50, year is interpreted as 20*yy*. For *yy* >= 50, year is interpreted as 19*yy*.	"4/15/94" "4.15.94" "4-15-94" "04.15/94"	All of these entries are interpreted as Apr 15 1994 when the dateformat option is set to mdy.
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 1-14 lists some examples of datetime entries and their display values:

Table 1-14: Examples of datetime entries

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

Finding *datetime* values that match a pattern

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

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

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

You can find the entry using the like operator:

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

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

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

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

```
like May 2%
```

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

Manipulating dates

You can do some arithmetic calculations on datetime values with the built-in date functions. See "Date functions".

Standards and compliance

Standard	Complience 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 datatypes, char(n) , and unichar (n) , and the variable-length datatypes, varchar(n) and univarchar (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 pagesize; 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) , unichar(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), univarchar(n), and nvarchar(n) . Data in variable-length columns is stripped of trailing blanks; storage size is the actual length of the data entered. Data in variable-length variables and parameters retains all trailing blanks, but is not padded to the defined length. Character literals are treated as variable-length datatypes.

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

Table 1-15: Character datatypes

Datatype	Stores	Bytes of Storage
char(n)	Fixed-length data, such as social security numbers or postal codes, in single-byte character sets.	*n*
unichar(n)	Fixed-length uncode data, in single-byte character sets.	*n*@@unicharsize (@@unicharsize equals 2)*
nchar(n)	Fixed-length data in multibyte character sets	*n * @@ncharsize*

Datatype	Stores	Bytes of Storage
varchar(n)	Variable-length data, such as names, in single-byte character sets.	Actual number of characters entered
univarchar(n)	Variable-length Unicode data, in single-byte character sets.	Actual number of characters * *@@unicharsize*
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, Adaptive Server stores it internally as a varchar.

If the min or max aggregate functions are used on a char column, the result returned is varchar, and is therefore stripped of all trailing spaces.

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.
- Only unichar not null columns are padded to the full width of the column; unichar null columns are treated like univarchar.
- Preceding blanks are not affected.
- Trailing blanks are truncated except for char, unichar and nchar not null columns.
- The empty string (“ ”) is treated as a single space. In char, *n*char and unichar 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	**Complience level**
SQL92	Transact-SQL provides the char and varchar SQL92 datatypes. The nchar, nvarchar, unichar, and univarchar datatypes are Transact-SQL extensions.

Binary datatypes

Function

Use the binary datatypes, binary(n) and varbinary(n), to store raw binary data, such as pictures, in a hexadecimal-like notation, up to the maximum column size for your server’s logical page size.

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.

Entries of more than the max column size

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

Treatment of trailing zeroes

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

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

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

```
insert zeros values (0x12345000, 0x12345000,
            0x12345000, 0x12345000)
insert zeros values (0x123, 0x123, 0x123, 0x123)
select * from zeros
bnot             bnull        vnot         vnull
------------     ---------    ----------  ---------
0x1234500000     0x123450     0x123450    0x123450
0x0123000000     0x0123       0x0123      0x0123
```

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

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

```
insert zeros values (0x0, 0x0,0x0, 0x0)
select * from zeros where bnot = 0x00
bnot           bnull       vnot        vnull
----------     ------      -----       ------------
0x0000000000   0x00        0x00        0x00
```

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

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

insert sample values ('002710000000ae1b')

select * from sample
col_a
------------------
0x3030323731303030
```

Platform dependence

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

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

For platform-independent conversions between hexadecimal strings and integers, use the inttohex and hextoint functions rather than the platform-specific convert function. For details, see “Datatype conversion functions”.

Standards and compliance

Standard	Complience level
SQL92	The binary and varbinary datatypes are Transact-SQL extensions.

bit datatype

Function

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

Entering data into *bit* columns

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

Storage size

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

Restrictions

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

Standards and compliance

Standard	Complience 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	Complience 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 that have an odd number of hexadecimal digits are padded with a leading zero (an insert of “0xaaabb” becomes “0x0aaabb”).

Effect of partitioning on data storage

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 text page has been allocated.

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

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

```
    create table imagetest
    (image_id varchar(6), imagecol image null,
    graphic_id char(4))
    insert imagetest values
    ("94732", 0x0000008300000000000100000000013c,
    "1389")
```

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

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

Saving space by allowing NULL

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 storage. For example, the following statement inserts values into the title_id and pub_id columns of the testtext table created above, but does not initialize the blurb text column:

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

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

Getting information from sysindexes

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

Table 1-16: Storage of text and image data

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

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

```
select name, data_pgs(object_id("blurbs"), ioampg)
from sysindexes
where name = "tblurbs"
name
------------------------------ -----------
tblurbs                                  7
```

Note The system tables poster shows a one-to-one (1-1) relationship between sysindexes and systabstats. This is correct, except for text and image columns, for which information is not kept in systabstats.

Using *readtext* and *writetext*

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

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

Determining how much space a column uses

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

```
sp_spaceused blurbs
```

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

Restrictions on *text* and *image* columns

text and image columns cannot be used:

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

Selecting *text* and *image* data

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

Table 1-17: text and image global variables

Variable	Explanation
@@textptr	The text pointer of the last text or image column inserted or updated by a process. Do not confuse this global variable with the 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 *text* and *image* datatypes

You can explicitly convert text values to char, unichar, varchar, and univarchar, 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, which is determined by the maximum column size for your server's logical page size. 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, univarchar, unichar 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 blurbs
where copy like "%Net Etiquette%"
```

Duplicate rows

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

Standards and compliance

Standard	Complience level
SQL92	The text and image datatypes are Transact-SQL extensions.

User-defined datatypes

Function

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

Creating frequently used datatypes in the *model* database

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

Creating a user-defined datatypes

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	Complience level
SQL92	User-defined datatypes are a Transact-SQL extension.

CHAPTER 2

Transact-SQL Functions

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

Types of functions

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

Table 2-1: Types of Transact-SQL functions

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

Table 2-2 lists the functions in alphabetical order.

Table 2-2: List of Transact-SQL functions

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

Function	Type	Return value
asin	Mathematical	The angle (in radians) whose sine is specified.
atan	Mathematical	The angle (in radians) whose tangent is specified.
atn2	Mathematical	The angle (in radians) whose sine and cosine are specified.
avg	Aggregate	The numeric average of all (distinct) values.
ceiling	Mathematical	The smallest integer greater than or equal to the specified value.
char	String	The character equivalent of an integer.
charindex	String	Returns an integer representing the starting position of an expression.
char_length	String	The number of characters in an expression.
col_length	System	The defined length of a column.
col_name	System	The name of the column whose table and column IDs are specified.
compare	System	Returns the following values, based on the collation rules that you chose: • 1 – indicates that *char_expression1* is greater than *char_expression2* • 0 – indicates that *char_expression1* is equal to *char_expression2* • -1 – indicates that *char_expression1* is less than *char_expression2*
convert	Datatype Conversion	The specified value, converted to another datatype or a different datetime display format.
cos	Mathematical	The cosine of the specified angle (in radians).
cot	Mathematical	The cotangent of the specified angle (in radians).
count	Aggregate	The number of (distinct) non-null values.
curunreservedpgs	System	The number of free pages in the specified disk piece.
data_pgs	System	The number of pages used by the specified table or index.
datalength	System	The actual length, in bytes, of the specified column or string.
dateadd	Date	The date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.
datediff	Date	The difference between two dates.
datename	Date	The name of the specified part of a datetime value.
datepart	Date	The integer value of the specified part of a datetime value.
db_id	System	The ID number of the specified database.
db_name	System	The name of the database whose ID number is specified.
degrees	Mathematical	The size, in degrees, of an angle with a specified number of radians.
difference	String	The difference between two soundex values.
exp	Mathematical	The value that results from raising the constant e to the specified power.
floor	Mathematical	The largest integer that is less than or equal to the specified value.

Function	Type	Return value
getdate	Date	The current system date and time.
hextoint	Datatype Conversion	The platform-independent integer equivalent of the specified hexadecimal string.
host_id	System	The host process ID of the client process.
host_name	System	The current host computer name of the client process.
index_col	System	The name of the indexed column in the specified table or view.
inttohex	Datatype Conversion	The platform-independent, hexadecimal equivalent of the specified integer.
isnull	System	Substitutes the value specified in *expression2* when *expression1* evaluates to NULL.
is_sec_service_on	Security	"1" if the security service is active; "0" if it is not.
isnull	String	The specified expression, trimmed of leading blanks.
lct_admin	System	Manages the last-chance threshold.
license_enabled	System	"1" if the feature's license is enabled; "0" if it is not.
log	Mathematical	The natural logarithm of the specified number.
log10	Mathematical	The base 10 logarithm of the specified number.
lower	String	The uppercase equivalent of the specified expression.
max	Aggregate	The highest value in a column.
min	Aggregate	The lowest value in a column.
mut_excl_roles	System	The mutual exclusivity between two roles.
object_id	System	The object ID of the specified object.
object_name	System	The name of the object whose object ID is specified.
patindex	String, Text and Image	The starting position of the first occurrence of a specified pattern.
pi	Mathematical	The constant value 3.1415926535897936.
power	Mathematical	The value that results from raising the specified number to a given power.
proc_role	System	1 if the user has the correct role to execute the procedure; 0 if the user does not have this role.
ptn_data_pgs	System	The number of data pages used by a partition.
radians	Mathematical	The size, in radians, of an angle with a specified number of degrees.
rand	Mathematical	A random value between 0 and 1, generated using the specified seed value.
replicate	String	A string consisting of the specified expression repeated a given number of times.
reserved_pgs	System	The number of pages allocated to the specified table or index.
reverse	String	The specified string, with characters listed in reverse order.

Function	Type	Return value
right	String	The part of the character expression, starting the specified number of characters from the right.
role_contain	System	1 if *role2* contains *role1*.
role_id	System	The system role ID of the role whose name you specify.
role_name	System	The name of a role whose system role ID you specify.
round	Mathematical	The value of the specified number, rounded to a given number of decimal places.
rowcnt	System	An estimate of the number of rows in the specified table.
rtrim	String	The specified expression, trimmed of trailing blanks.
show_role	System	The login's currently active roles.
show_sec_services	Security	A list of the user's currently active security services.
sign	Mathematical	The sign (+1 for positive, 0, or -1 for negative) of the specified value.
sin	Mathematical	The sine of the specified angle (in radians).
sortkey	System	Values that can be used to order results based on collation behavior, which allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity.
soundex	String	A 4-character code representing the way an expression sounds.
space	String	A string consisting of the specified number of single-byte spaces.
sqrt	Mathematical	The square root of the specified number.
str	String	The character equivalent of the specified number.
stuff	String	The string formed by deleting a specified number of characters from one string and replacing them with another string.
substring	String	The string formed by extracting a specified number of characters from another string.
sum	Aggregate	The total of the values.
suser_id	System	The server user's ID number from the syslogins system table.
suser_name	System	The name of the current server user, or the user whose server user ID is specified.
syb_sendmsg		Sends a message to a User Datagram Protocol (UDP) port.
tan	Mathematical	The tangent of the specified angle (in radians).
textptr	Text and Image	The pointer to the first page of the specified text column.
textvalid	Text and Image	1 if the pointer to the specified text column is valid; 0 if it is not.
to_unichar	String	A unichar expression having the value of the integer expression.
tsequal	System	Compares timestamp values to prevent update on a row that has been modified since it was selected for browsing.
uhighsurr	String	1 if the Unicode value at position start is the high half of a surrogate pair (which should appear first in the pair); otherwise 0.

Function	Type	Return value
ulowsurr	String	1 if the Unicode value at position start is the low half of a surrogate pair (which should appear second in the pair); otherwise 0.
upper	String	The uppercase equivalent of the specified string.
uscalar	String	The Unicode scalar value for the first Unicode character in an expression.
used_pgs	System	The number of pages used by the specified table and its clustered index.
user	System	The name of the current server user.
user_id	System	The ID number of the specified user or the current user.
user_name	System	The name within the database of the specified user or the current user.
valid_name	System	0 if the specified string is not a valid identifier; a number other than 0 if the string is valid.
valid_user	System	1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

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

Aggregate functions

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

- avg
- count
- max
- min
- sum

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

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

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

The max, min, and count aggregate functions now have semantics that include the unichar data type.

Aggregates used with *group by*

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

Aggregate functions and NULL values

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

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

Vector and scalar aggregates

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

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

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

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

(6 rows affected)
```

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

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

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

Example 1

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



type
------------  ------------  ------------
UNDECIDED             NULL          NULL
business             13.73      6,281.25
mod_cook             11.49      7,500.00
popular_comp         21.48      7,500.00
psychology           13.50      4,255.00
trad_cook            15.96      6,333.33

(6 rows affected)
```

Example 2

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

type         title_id
-----------  --------    ---------- ---------
UNDECIDED       MC3026      NULL        NULL
```

```
business        BU1032      13.73   6,281.25
business        BU1111      13.73   6,281.25
business        BU2075      13.73   6,281.25
business        BU7832      13.73   6,281.25
mod_cook        MC2222      11.49   7,500.00
mod_cook        MC3021      11.49   7,500.00
popular_comp    PC1035      21.48   7,500.00
popular_comp    PC8888      21.48   7,500.00
popular_comp    PC9999      21.48   7,500.00
psychology      PS1372      13.50   4,255.00
psychology      PS2091      13.50   4,255.00
psychology      PS2106      13.50   4,255.00
psychology      PS3333      13.50   4,255.00
psychology      PS7777      13.50   4,255.00
trad_cook       TC3218      15.96   6,333.33
trad_cook       TC4203      15.96   6,333.33
trad_cook       TC7777      15.96   6,333.33
```

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

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

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

Example 3

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

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

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

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

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

Aggregate functions as row aggregates

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

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

Start of select statement

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

where:

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

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

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

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

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

The following examples illustrate the differences:

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

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

(2 rows affected)
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
type        price       advance
----------  ----------  ---------------
mod_cook          2.99        15,000.00
mod_cook         19.99             0.00
            sum         sum
            ----------  ---------------
                 22.98        15,000.00
type         price          advance
----------  ----------  ---------------
trad_cook        11.95         4,000.00
trad_cook        14.99         8,000.00
trad_cook        20.95         7,000.00
            sum         sum
            ----------  ---------------
                 47.89        19,000.00
(7 rows affected)
type        price       advance
----------  ----------  ---------------
mod_cook          2.99        15,000.00
mod_cook         19.99             0.00

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

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

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

The order of columns in the select list overrides the order of the aggregates in the compute clause. For example:

```
create table t1 (a int, b int, c int null)
insert t1 values(1,5,8)
insert t1 values(2,6,9)
(1 row affected)

compute sum(c),  max(b), min(a)
select a, b, c from t1
 a           b           c
 ----------- ----------- -----------
           1           5           8
           2           6           9

Compute Result:
 ----------- ----------- -----------
           1           6          17
```

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

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

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

Datatype conversion functions

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

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

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

Adaptive Server performs certain datatype conversions automatically. These are called *implicit conversions*. For example, if you compare a char expression and a datetime expression, or a smallint expression and an int expression, or char expressions of different lengths, Adaptive Server automatically converts one datatype to another.

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

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

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

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

From: To:	tinyint	smallint	int	decimal	numeric	real	float	[n]char	[n]varchar	unichar	univarchar	text	smallmoney	money	bit	smalldatetime	datetime	binary	varbinary	image
tinyint	–	I	I	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
smallint	I	–	I	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
int	I	I	–	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
decimal	I	I	I	I/E	I/E	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
numeric	I	I	I	I/E	I/E	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
real	I	I	I	I	I	–	I	E	E	E	E	U	I	I	I	U	U	I	I	U
float	I	I	I	I	I	I	–	E	E	E	E	U	I	I	I	U	U	I	I	U
[n]char	E	E	E	E	E	E	E	I	I	I	I	I	E	E	E	I	I	I	I	I
[n]varchar	E	E	E	E	E	E	E	I	I	I	I	I	E	E	E	I	I	I	I	I
unichar	E	E	E	E	E	E	E	I	I	–	I	I	E	E	E	E	E	I	I	I
univarchar	E	E	E	E	E	E	E	I	I	I	–	I	E	E	E	E	E	I	I	I
text	U	U	U	U	U	U	U	E	E	E	E	U	U	U	U	U	U	U	U	U
smallmoney	I	I	I	I	I	I	I	I	I	E	E	U	–	I	I	U	U	I	I	U
money	I	I	I	I	I	I	I	I	I	E	E	U	I	–	I	U	U	I	I	U
bit	I	I	I	I	I	I	I	I	I	E	E	U	I	I	–	U	U	I	I	U
smalldatetime	U	U	U	U	U	U	U	I	I	E	E	U	U	U	U	–	I	I	I	U
datetime	U	U	U	U	U	U	U	I	I	E	E	U	U	U	U	I	–	I	I	U

binary	I	I	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	–	I	I
varbinary	I	I	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I	–	I
image	U	U	U	U	U	U	U	U	U	E	E	U	U	U	U	U	U	E	E	U

Key:

E Explicit datatype conversion is required.

I Conversion can be done either implicitly or with an explicit datatype conversion function.

I/E Explicit datatype conversion function required when there is loss of precision or scale and arithabort numeric_truncation is on; otherwise, implicit conversion is allowed.

U Unsupported conversion.

– Conversion of a datatype to itself. These conversions are allowed but are meaningless.

Converting character data to a non-character type

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

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

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

Converting from one character type to another

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

text columns can be explicitly converted to char, nchar, varchar, unichar, univarchar, or nvarchar. You are limited to the maximum length of the character datatypes, which is determined by the maximum column size for your server's logical page size. If you do not specify the length, the converted value has a default length of 30 bytes.

Converting numbers to a character type

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

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

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

Rounding during conversion to and from money types

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

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

```
select convert(int, $4.50)
 -----------
           5
```

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

Converting date/time information

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

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

Converting between numeric types

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

For example, if you provide a float or numeric value as an argument to a built-in function that expects an integer, the value of the float or numeric is truncated. However, Adaptive Server does not implicitly convert numerics that have a fractional part but returns a scale error message. For example, Adaptive Server returns error 241 for numerics that have a fractional part and error 257 if other datatypes are passed.

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

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

Arithmetic overflow and divide-by-zero errors

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

- Explicit or implicit conversions to exact types with a lower precision or scale

- Explicit or implicit conversions of data that falls outside the acceptable range for a money or date/time type
- Conversions of hexadecimal strings requiring more than 4 bytes of storage using hextoint

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

Use the arithignore arith_overflow option to determine whether Adaptive Server displays a message after these errors. The default setting, off, displays a warning message when a divide-by-zero error or a loss of precision occurs. Setting arithignore arith_overflow on suppresses warning messages after these errors. The optional arith_overflow keyword can be omitted without any effect.

Scale errors

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

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

Note For entry level SQL92 compliance, set:

- arithabort arith_overflow off
- arithabort numeric_truncation on
- arithignore off

Domain errors

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

Conversions between binary and integer types

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

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

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

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

Converting between binary and numeric or decimal types

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

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

```
select convert(numeric
(38, 18),0x000000000000000006b14bd1e6eea0000000000000000000000000000000)



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

This example converts the same numeric data back to binary:

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



--------------------------------------------------------------
0x000000000000000006b14bd1e6eea0000000000000000000000000000000
```

Converting image columns to binary types

You can use the convert function to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes, which is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 characters.

Converting other types to *bit*

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

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

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

Converting NULL value

You can use the convert function to change the NULL to NOT NULL and NOT NULL to NULL.

Date functions

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

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

Use the datetime datatype only for dates after January 1, 1753. datetime values must be enclosed in single or double quotes. Use char, nchar, varchar or nvarchar for earlier dates. Adaptive Server recognizes a wide variety of date formats. See Datatype conversion functions and “Date and time datatypes” for more information.

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

Date parts

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

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

When you enter a year as two digits (*yy*):

- Numbers less than 50 are interpreted as 20*yy*. For example, `01` is 2001, `32` is 2032, and `49` is 2049.
- Numbers equal to or greater than 50 are interpreted as 19*yy*. For example, `50` is 1950, `74` is 1974, and `99` is 1999.

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

Mathematical functions

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

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

The mathematical functions are:

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

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

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

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

- By default, the arithignore arith_overflow option is turned off, causing Adaptive Server to display a warning message after any query that results in numeric overflow. Set the arithignore option on to ignore overflow errors.

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

Security functions

Security functions return security-related information.

The security functions are:

- is_sec_service_on
- show_sec_services

String functions

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

- ascii
- char
- charindex
- char_length
- difference
- lower
- ltrim
- patindex
- replicate

- reverse
- right
- rtrim
- soundex
- space
- str
- stuff
- substring
- to_unichar
- uhighsurr
- ulowsurr
- upper
- uscalar

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

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

When a string function accepts two character expressions but only one expression is unichar, the other expression is "promoted" and internally converted to unichar. This follows existing rules for mixed-mode expressions. However, this conversion may cause truncation, since unichar data sometimes takes twice the space.

Limits on string functions

Results of string functions are limited to 16K .

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

```
select replicate("a", 900) + replicate("B", 900)
```

Displays the first 16K of data, but the subsequent data is not displayed.

System functions

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

- col_length
- col_name
- curunreservedpgs
- data_pgs
- datalength
- db_id
- db_name
- host_id
- host_name
- index_col
- isnull
- lct_admin
- mut_excl_roles
- object_id
- object_name
- proc_role
- ptn_data_pgs
- reserved_pgs
- role_contain
- role_id
- role_name

- rowcnt
- show_role
- suser_id
- suser_name
- tsequal
- used_pgs
- user
- user_id
- user_name
- valid_name
- valid_user

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

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

Text and image functions

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

- textptr
- textvalid

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

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

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

text and image columns cannot be used:

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

CHAPTER 3 Functions: *abs – difference*

abs

Description

Returns the absolute value of an expression.

Syntax

abs(*numeric_expression*)

Parameters

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

Examples

```
select abs(-1)

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

Returns the absolute value of -1.

Usage

- abs, a mathematical function, returns the absolute value of a given expression. Results are of the same type and have the same precision and scale as the numeric expression.
- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute abs.

See also

Functions – ceiling, floor, round, sign

acos

Description

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

Syntax

acos(*cosine*)

Parameters

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

Examples

```
select acos(0.52)

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

Returns the angle whose cosine is 0.52.

Usage

- acos, a mathematical function, returns the angle (in radians) whose cosine is the specified value.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute acos.

See also

Functions – cos, degrees, radians

ascii

Description

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

Syntax

ascii(*char_expr*|*uchar_expr*)

Parameters

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

uchar_expr
– is a character-type column name, variable, or constant expression of unichar or univarchar type.

Examples

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

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

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

Usage

- ascii, a string function, returns the ASCII code for the first character in the expression.
- When a string function accepts two character expressions but only one expression is unichar, the other expression is “promoted” and internally converted to unichar. This follows existing rules for mixed-mode expressions. However, this conversion may cause truncation, since unichar data sometimes takes twice the space.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute ascii.

See also

Functions – char, to_unichar

asin

Description

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

Syntax

asin(*sine*)

Parameters

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

Examples

```
select asin(0.52)

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

Usage

- asin, a mathematical function, returns the angle (in radians) whose sine is the specified value.
- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute asin.

See also — *Functions* – degrees, radians, sin

atan

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

Syntax — atan(*tangent*)

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

Examples

```
select atan(0.50)

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

Usage

- atan, a mathematical function, returns the angle (in radians) whose tangent is the specified value.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute atan.

See also — *Functions* – atn2, degrees, radians, tan

atn2

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

Syntax — atn2(*sine*, *cosine*)

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

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

Examples

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

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

Usage

- atn2, a mathematical function, returns the angle (in radians) whose sine and cosine are specified.
- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute atn2.

See also: *Functions* – atan, degrees, radians, tan

avg

Description: Returns the numeric average of all (distinct) values.

Syntax: avg([all | distinct] *expression*)

Parameters

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

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

expression
– is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see “Expressions” on page 179.

Examples

Example 1

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

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

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

Example 2

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


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

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

Example 3

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

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

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

Usage

- avg, an aggregate function, finds the average of the values in a column. avg can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating averages.

- For general information about aggregate functions, see “Aggregate functions” on page 45.
- When you average integer data, Adaptive Server treats the result as an int value, even if the datatype of the column is smallint or tinyint. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type int.
- You cannot use avg() with the binary datatypes.
- Since the average value is only defined on numeric datatypes, use with Unicode expressions generates an error.

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute avg.

See also: *Functions* – max, min

ceiling

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

Syntax: ceiling(*value*)

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

Examples: **Example 1**

```
select ceiling(123.45)
```

```
124
```

Example 2

```
select ceiling(-123.45)
```

```
-123
```

Example 3

```
select ceiling(1.2345E2)
```

```
24.000000
```

Example 4

```
select ceiling(-1.2345E2)
```

```
-123.000000
```

Example 5

```
select ceiling($123.45)
```

```
124.00
```

Example 6

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

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

Usage

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

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

- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute ceiling.

See also

Command – set

Functions – abs, floor, round, sign

char

Description

Returns the character equivalent of an integer.

Syntax

char(*integer_expr*)

Parameters

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

Examples

Example 1

```
select char(42)

-
*
```

Example 2

```
select xxx = char(65)

xxx
---
A
```

Usage

- char, a string function, converts a single-byte integer value to a character value. (char is usually used as the inverse of ascii.)
- char returns a char datatype. If the resulting value is the first byte of a multibyte character, the character may be undefined.
- If *char_expr* is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 62.

Reformatting output with char

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

  For example:

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



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

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

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute char.

See also *Functions* – ascii, str

charindex

Description Returns an integer representing the starting position of an expression.

Syntax charindex(*expression1*, *expression2*)

Parameters *expression*

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

Examples

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

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

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

Usage

- charindex, a string function, searches *expression2* for the first occurrence of *expression1* and returns an integer representing its starting position. If *expression1* is not found, charindex returns 0.
- If *expression1* contains wildcard characters, charindex treats them as literals.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- If a varchar expression is given as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).
- For general information about string functions, see “String functions” on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute charindex.

See also
Function – patindex

char_length

Description
Returns the number of characters in an expression.

Syntax
char_length(*char_expr*|*uchar_expr*)

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

uchar_expr
– is a character-type column name, variable, or constant expression of unichar or univarchar type.

Examples
Example 1

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

 -----------
          39
```

Example 2

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

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

Usage
- char_length, a string function, returns an integer representing the number of characters in a character expression or text value.

- For variable-length columns and variables, char_length returns the number of characters (not the defined length of the column or variable). If explicit trailing blanks are included in variable-length variables, they are not stripped. For literals and fixed-length character columns and variables, char_length does not strip the expression of trailing blanks (see example 2).
- For multi-byte character sets, the number of characters in the expression is usually less than the number of bytes; use datalength to determine the number of bytes.
- For Unicode expressions, returns the number of Unicode values (not bytes) in an expression. Surrogate pairs count as two Unicode values.
- If *char_expr* or *uchar_expr* is NULL, char_length returns NULL.
- For general information about string functions, see "String functions" on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute char_length.

See also

Function – datalength

col_length

Description

Returns the defined length of a column.

Syntax

col_length(*object_name*, *column_name*)

Parameters

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

column_name
– is the name of the column.

Examples

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

 x
 ----
   80
```

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

Usage

- col_length, a system function, returns the defined length of column.
- For general information about system functions, see “System functions” on page 64.
- To find the actual length of the data stored in each row, use datalength.
- For text and image columns, col_length returns 16, the length of the binary(16) pointer to the actual text page.
- For unichar columns, the defined length is the number of Unicode values declared when the column was defined (not the number of bytes represented).

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute col_length.

See also

Function – datalength

col_name

Description

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

Syntax

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

Parameters

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

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

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

Examples

```
select col_name(208003772, 2)

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

Usage

- col_name, a system function, returns the column’s name.
- For general information about system functions, see “System functions” on page 64.

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute col_name.

See also *Functions* – db_id, object_id

compare

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

Syntax

```
compare (char_expression1|uchar_expression1),
        (char_expression2|uchar_expression2)
         [,{collation_name | collation_ID}]
```

Parameters

char_expression1 or uchar_expression 1
– are the character expressions you want to compare to *char_expression2* or *uchar_expression 2*.

char_expression2 or *uchar_expression2*
– are the character expressions against which you want to compare *char_expression1* or *uchar_expression1*..

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

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

uchar_expression1 and *uchar_expression2* can be one of the following:

- Character type (unichar or univarchar)
- Character variable, or
- Constant character expression, enclosed in single or double quotation marks

collation_name
– can be a quoted string or a character variable that specifies the collation to use. Table 3-1 shows the valid values.

collation_ID
– is an integer constant or a variable that specifies the collation to use. Table 3-1 shows the valid values.

Usage

- The compare function returns the following values, based on the collation rules that you chose:
    - 1 – indicates that *char_expression1* or *uchar_expression1* is greater than *char_expression2* or *uchar_expression2*.
    - 0 – indicates that *char_expression1* or *uchar_expression1* is equal to *char_expression2* or *uchar_expression2*.
    - -1 – indicates that *char_expression1* or *uchar_expression1* is less than *char_expression2* or *uchar expression2*.
- Both *char_expression1*, *uchar_expression1*, and *char_expression2* and *uchar_expression2* must be characters that are encoded in the server's default character set.
- Either *char_expression1*, *uchar_expression 1*, or *char_expression2*, *uchar_expression2*, or both, can be empty strings:
    - If *char_expression2* or *uchar_expression2* is empty, the function returns 1.
    - If both strings are empty, then they are equal, and the function returns a 0 value.
    - If *char_expression1* or *uchar_expression 1* is empty, the function returns a -1.

  The compare function does not equate empty strings and strings containing only spaces, as does. compare uses the sortkey function to generate collation keys for comparison. Therefore, a truly empty string, a string with one space, or a string with two spaces will not compare equally.
- If either *char_expression1*, *uchar_expression1;* or *char_expression2*, *uchar_expression2* is NULL, then the result will be NULL.
- If a varchar expression is given as one parameter and a unichar expression is given as the other, the varchar expression is implicitly converted to unichar (with possible truncation).
- If you do not specify a value for *collation_name* or *collation_ID*, compare assumes binary collation.
- Table 3-1 lists the valid values for *collation_name* and *collation_ID*.

Table 3-1: Collation names and IDs

Description	Collation name	Collation ID
Binary sort	binary	50

Description	Collation name	Collation ID
Default Unicode multilingual	default	0
CP 850 Alternative no accent	altnoacc	39
CP 850 Alternative lower case first	altdict	45
CP 850 Alternative no case preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian no case preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-9 Turkish dictionary	turdict	72
Shift-JIS binary order	sjisbin	259
Thai dictionary	thaidict	1

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute compare.

See also *Function* – sortkey

convert

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

Syntax

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

Parameters

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

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

length
– is an optional parameter used with char, nchar, unichar, univarchar, varchar, nvarchar, binary and varbinary datatypes. If you do not supply a length, Adaptive Server truncates the data to 30 characters for the character types and 30 bytes for the binary types. The maximum allowable length for character and binary expression is 64K.

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

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

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

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

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

When Unichar is used as the destination data type, the default length of 30 Unicode values is used if no length is specified.

style

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

When converting datetime or smalldatetime data to a character type, use the style numbers in Table 3-2 to specify the display format. Values in the left-most column display 2-digit years (yy). For 4-digit years (yyyy), add 100, or use the value in the middle column.

Table 3-2: Display formats for date/time information

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

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

Examples

Example 1

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

Example 2

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

Example 3

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

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

Example 4

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

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

Example 5

```
select convert(integer, 0x00000100)
```

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

Example 6

```
select convert (binary, 10)
```

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

Example 7

```
select convert(bit, $1.11)
```

Returns 1, the bit string equivalent of $1.11.

Example 8

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

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

Usage

- convert, a datatype conversion function, converts between a wide variety of datatypes and reformats date/time and money data for display purposes.
- For more information about datatype conversion, see “Datatype conversion functions” on page 51.
- convert() generates a domain error when the argument falls outside the range over which the function is defined. This should happen rarely.
- Use null or not null to specify the nullability of a target column. Specifically, this can be used with select into to create a new table and change the datatype and nullability of existing columns in the source table (See example 8, above).

- You can use convert to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes, which is determined by the maximum column size for your server's logical page size. If you do not specify the length, the converted value has a default length of 30 characters.
- Unichar expressions can be used as a destination data type or they can be converted to another data type. Unichar expresssions can be converted either explicitly between any other data type supported by the server, or implicitly.
- If length is not specifed when unichar is used as a destination type, the default length of 30 Unicode values is used. If the length of the destination type is not large enough to accommodate the given expression, as error message appears.

Conversions involving Java classes

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

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

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute convert.

See also

Datatypes – User-defined datatypes

Functions – hextoint, inttohex

cos

Description

Returns the cosine of the specified angle.

Syntax

cos(*angle*)

Parameters

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

Examples

```
select cos(44)

 0.999843
```

Usage

- cos, a mathematical function, returns the cosine of the specified angle (in radians).
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute cos.

See also

Functions – acos, degrees, radians, sin

cot

Description

Returns the cotangent of the specified angle.

Syntax

cot(*angle*)

Parameters

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

Examples

```
select cot(90)

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

Usage

- cot, a mathematical function, returns the cotangent of the specified angle (in radians).
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute cot.

See also

Functions – degrees, radians, sin

count

Description

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

Syntax

count([all | distinct] *expression*)

Parameters

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

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

expression
is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" on page 179.

Examples

Example 1

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

Finds the number of different cities in which authors live.

Example 2

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

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

Usage

- count, an aggregate function, finds the number of non-null values in a column. For general information about aggregate functions, see "Aggregate functions" on page 45.
- When distinct is specified, count finds the number of unique non-null values. count can be used with all datatypes, including unichar, but cannot be used with text and image. Null values are ignored when counting.
- count(*column_name*) returns a value of 0 on empty tables, on columns that contain only null values, and on groups that contain only null values.

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

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

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

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

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute count.

See also

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

curunreservedpgs

Description

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

Syntax

curunreservedpgs(*dbid*, *lstart*, *unreservedpgs*)

Parameters

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

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

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

Examples

Example 1

```
select db_name(dbid), d.name,
      curunreservedpgs(dbid, lstart, unreservedpgs)
```

```
            from sysusages u, sysdevices d
            where d.low <= u.size + vstart
                and d.high >= u.size + vstart -1
                and d.status &2 = 2

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

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

Example 2

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

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

Usage

- curunreservedpgs, a system function, returns the number of free pages in a disk piece. For general information about system functions, see "System functions" on page 64.
- If the database is open, the value is taken from memory; if the database is not in use, the value is taken from the *unreservedpgs* column in *sysusages*.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute curunreservedpgs.

See also

Functions – db_id, lct_admin

data_pgs

Description

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

Syntax

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

Parameters

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

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

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

Examples

Example 1

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

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

Example 2

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

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

Usage

- data_pgs, a system function, returns the number of pages used by a table (*doampg*) or index (*ioampg*). You must use this function in a query run against the sysindexes table. For more information on system functions, see “System functions” on page 64.
- data_pgs works only on objects in the current database.
- The result does not include pages used for internal structures. To see a report of the number of pages for the table, clustered index, and internal structures, use used_pgs.

Accuracy of results

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

Errors

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

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute data_pgs.

See also: *Functions* – object_id, rowcnt

System procedure – sp_spaceused

datalength

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

Syntax: datalength(*expression*)

Parameters: *expression*

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

Examples:

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

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

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

Usage

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

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute datalength.

See also

Functions – char_length, col_length

dateadd

Description

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

Syntax

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

Parameters

date_part
– is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see “Date parts” on page 59.

numeric
– is an integer expression.

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

Examples

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

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

Usage

- dateadd, a date function, adds an interval to a specified date. For more information about date functions, see "Date functions" on page 59.
- dateadd takes three arguments—the date part, a number, and a date. The result is a datetime value equal to the date plus the number of date parts.

  If the date argument is a smalldatetime value, the result is also a smalldatetime. You can use dateadd to add seconds or milliseconds to a smalldatetime, but it is meaningful only if the result date returned by dateadd changes by at least one minute.
- Use the datetime datatype only for dates after January 1, 1753. datetime values must be enclosed in single or double quotes. Use char, nchar, varchar or nvarchar for earlier dates. Adaptive Server recognizes a wide variety of date formats. For more information, see "User-defined datatypes" on page 38 and "Datatype conversion functions" on page 51.

  Adaptive Server automatically converts between character and datetime values when necessary (for example, when you compare a character value to a datetime value).
- Using the date part weekday or dw with dateadd is not logical, and produces spurious results. Use day or dd instead.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute dateadd.

See also

Datatypes – Date and time datatypes

Commands – select, where Clause

Functions – datediff, datename, datepart, getdate

datediff

Description

Returns the difference between two dates.

Syntax

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

Parameters

datepart
– is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see "Date parts" on page 59.

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

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

Examples

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

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

Usage

- datediff, a date function, calculates the number of date parts between two specified dates. For more information about date functions, see "Date functions" on page 59.
- datediff takes three arguments. The first is a date part. The second and third are dates. The result is a signed integer value equal to *date2* - *date1*, in date parts.
- datediff produces results of datatype int, and causes errors if the result is greater than 2,147,483,647. For milliseconds, this is approximately 24 days, 20:31.846 hours. For seconds, this is 68 years, 19 days, 3:14:07 hours.
- datediff results are always truncated, not rounded, when the result is not an even multiple of the date part. For example, using hour as the date part, the difference between "4:00AM" and "5:50AM" is 1.

  When you use day as the date part, datediff counts the number of midnights between the two times specified. For example, the difference between January 1, 1992, 23:00 and January 2, 1992, 01:00 is 1; the difference between January 1, 1992 00:00 and January 1, 1992, 23:59 is 0.
- The month datepart counts the number of first-of-the-months between two dates. For example, the difference between January 25 and February 2 is 1; the difference between January 1 and January 31 is 0.
- When you use the date part week with datediff, you get the number of Sundays between the two dates, including the second date but not the first. For example, the number of weeks between Sunday, January 4 and Sunday, January 11 is 1.

- If smalldatetime values are used, they are converted to datetime values internally for the calculation. Seconds and milliseconds in smalldatetime values are automatically set to 0 for the purpose of the difference calculation.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute datediff.

See also

Datatypes – Date and time datatypes

Commands – select, where Clause

Functions – dateadd, datename, datepart, getdate

datename

Description

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

Syntax

datename (*datepart*, *date*)

Parameters

datepart
– is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see "Date parts" on page 59.

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

Examples

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

```
November
```

This example assumes a current date of November 20, 2000.

Usage

- datename, a date function, returns the name of the specified part (such as the month "June") of a datetime or smalldatetime value, as a character string. If the result is numeric, such as "23" for the day, it is still returned as a character string.
- For more information about date functions, see "Date functions" on page 59.
- The date part weekday or dw returns the day of the week (Sunday, Monday, and so on) when used with datename.

- Since smalldatetime is accurate only to the minute, when a smalldatetime value is used with datename, seconds and milliseconds are always 0.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute datename.

See also
Datatypes – Date and time datatypes \

Commands – select, where Clause

Functions – dateadd, datename, datepart, getdate

datepart

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

Syntax
datepart(*date_part*, *date*)

Parameters
date_part
– is a date part. Table 3-3 lists the date parts, the abbreviations recognized by datepart, and the acceptable values.

Table 3-3: Date parts and their values

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

When you enter a year as two digits (*yy*):

- Numbers less than 50 are interpreted as 20*yy*. For example, `01` is 2001, `32` is 2032, and `49` is 2049.
- Numbers equal to or greater than 50 are interpreted as 19*yy*. For example, `50` is 1950, `74` is 1974, and `99` is 1999.

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

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

Examples

Example 1

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

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

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

Example 2

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

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

Example 3

```
select datepart(cwk,'1993/01/01')

-----------
          53
```

Example 4

```
select datepart(cyr,'1993/01/01')

-----------
        1992
```

Example 5

```
select datepart(cdw,'1993/01/01')

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

Usage

- datepart, a date function, returns an integer value for the specified part of a datetime value. For more information about date functions, see “Date functions” on page 59.
- datepart returns a number that follows ISO standard 8601, which defines the first day of the week and the first week of the year. Depending on whether the datepart function includes a value for calweekofyear, calyearofweek, or caldayorweek, the date returned may be different for the same unit of time. For example, if Adaptive Server is configured to use US English as the default language:

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

  returns 1988, but:

  ```
  datepart(yy, "1/1/1989)
  ```

  returns 1989.

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

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

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

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute datepart.

See also

Datatypes – Date and time datatypes

Commands – select, where Clause

Functions – dateadd, datediff, datename, getdate

db_id

Description

Returns the ID number of the specified database.

Syntax

db_id(*database_name*)

Parameters

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

Examples

```
select db_id("sybsystemprocs")

------
4
```

Usage

- db_id, a system function, returns the database ID number.
- If you do not specify a *database_name*, db_id returns the ID number of the current database.
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute db_id.

See also

Functions – db_name, object_id

db_name

Description

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

Syntax

db_name([*database_id*])

Parameters

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

Examples

Example 1

```
select db_name()
```

Returns the name of the current database.

Example 2

```
select db_name(4)


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

Usage

- db_name, a system function, returns the database name.

- If no *database_id* is supplied, db_name returns the name of the current database.
- For general information about system functions, see “System functions” on page 64.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute db_name.

See also
Functions – col_name, db_id, object_name

degrees

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

Syntax
degrees(*numeric*)

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

Examples

```
select degrees(45)

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

Usage

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

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

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

- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute degrees.

See also
Functions – radians

difference

Description

Returns the difference between two soundex values.

Syntax

difference(*char_expr1*|u*char_expr1*), (*char_expr2*| *uchar_expr2*)

Parameters

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

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

uchar_expr1
– is a character-type column name, variable, or constant expression of unichar type.

uchar_expr2
– is another character-type column name, variable, or constant expresssion of unichar type.

Examples

Example 1

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

---------
4
```

Example 2

```
select difference("smothers", "brothers")

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

Usage

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

  The string values must be composed of a contiguous sequence of valid single- or double-byte roman letters.
- If *char_expr1*, *uchar_expr1*, or *char_expr2*, *uchar_expr2* is NULL, returns NULL.

- If a varchar expression is given as one parameter and a unichar expression is given as the other, the varchar expression is implicitly converted to unichar (with possible truncation).
- For general information about string functions, see “String functions” on page 62.

Standards	SQL92 – Complience level: Transact-SQL extension
Permissions	Any user can execute difference.
See also	*Functions* – soundex

CHAPTER 4

Functions: *exp – mut_excl_roles*

exp

Description	Returns the value that results from raising the constant e to the specified power.
Syntax	exp(*approx_numeric*)
Parameters	*approx_numeric* – is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

Examples

```
select exp(3)

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

Usage

- exp, a mathematical function, returns the exponential value of the specified value.
- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards	SQL92 – Complience level: Transact-SQL extension
Permissions	Any user can execute exp.
See also	*Functions* – log, log10, power

floor

Description	Returns the largest integer that is less than or equal to the specified value.
Syntax	floor(*numeric*)
Parameters	*numeric* – is any exact numeric (numeric, dec, decimal, tinyint, smallint, or int), approximate numeric (float, real, or double precision), or money column, variable, constant expression, or a combination of these.

Examples

Example 1

```
select floor(123)

-----------
        123
```

Example 2

```
select floor(123.45)

-------
    123
```

Example 3

```
select floor(1.2345E2)

--------------------
         123.000000
```

Example 4

```
select floor(-123.45)

-------
   -124
```

Example 5

```
select floor(-1.2345E2)

--------------------
        -124.000000
```

Example 6

```
select floor($123.45)

------------------------
                 123.00
```

Usage

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

  For numeric and decimal expressions, the results have a precision equal to that of the expression and a scale of 0.

- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute floor.

See also *Functions* – abs, ceiling, round, sign

getdate

Description Returns the current system date and time.

Syntax getdate()

Parameters None.

Examples **Example 1**

```
select getdate()
```

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

Example 2

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

```
1
```

Example 3

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

```
November
```

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

Usage

- getdate, a date function, returns the current system date and time.
- For more information about date functions, see “Date functions” on page 59.

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute getdate.

See also *Datatypes* – Date and time datatypes

Functions – dateadd, datediff, datename, datepart

hextoint

Description	Returns the platform-independent integer equivalent of a hexadecimal string.
Syntax	hextoint (*hexadecimal_string*)
Parameters	*hexadecimal_string* – is the hexadecimal value to be converted to an integer. This must be either a character type column or variable name or a valid hexadecimal string, with or without a "0x" prefix, enclosed in quotes.

Examples

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

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

Usage

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

  hextoint returns the integer equivalent of the hexadecimal string. The function always returns the same integer equivalent for a given hexadecimal string, regardless of the platform on which it is executed.
- For more information about datatype conversion, see "Datatype conversion functions" on page 51.

Standards	SQL92 – Complience level: Transact-SQL extension
Permissions	Any user can execute hextoint.
See also	*Functions* – convert, inttohex

host_id

Description	Returns the host process ID or the client process.
Syntax	host_id()
Parameters	None.

Examples

```
select host_id()

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

Usage

- host_id, a system function, returns the host process ID of the client process (not the Server process).
- For general information about system functions, see “String functions” on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute host_id.

See also

Function – host_name

host_name

Description

Returns the current host computer name of the client process.

Syntax

host_name()

Parameters

None.

Examples

```
select host_name()

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

Usage

- host_name, a system function, returns the current host computer name of the client process (not the Server process).
- For general information about system functions, see “System functions” on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute host_name.

See also

Function – host_id

index_col

Description

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

Syntax

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

Parameters

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

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

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

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

Examples

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

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

Usage

- index_col, a system function, returns the name of the indexed column.
- index_col returns NULL if *object_name* is not a table or view name.
- For general information about system functions, see "String functions" on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute index_col.

See also

Functions – object_id

System procedures – sp_helpindex

index_colorder

Description

Returns the column order.

Syntax

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

Parameters

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

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

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

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

Examples

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

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

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

Usage

- index_colorder, a system function, returns "ASC" for columns in ascending order or "DESC" for columns in descending order.
- index_colorder returns NULL if *object_name* is not a table name or if *key_#* is not a valid key number.
- For general information about system functions, see "String functions" on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute index_colorder.

inttohex

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

Syntax
: inttohex (*integer_expression*)

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

Examples

```
select inttohex (10)

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

Usage

- inttohex, a datatype conversion function, returns the platform-independent hexadecimal equivalent of an integer, without a "0x" prefix.
- Use the inttohex function for platform-independent conversions of integers to hexadecimal strings. inttohex accepts any expression that evaluates to an integer. It always returns the same hexadecimal equivalent for a given expression, regardless of the platform on which it is executed.
- For more information about datatype conversion, see "Datatype conversion functions" on page 51.

Standards
: SQL92 – Complience level: Transact-SQL extension

Permissions
: Any user can execute inttohex.

See also
: *Functions* – convert, hextoint

isnull

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

Syntax
: isnull(*expression1*, *expression2*)

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

Examples

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

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

Usage

- isnull, a system function, substitutes the value specified in *expression2* when *expression1* evaluates to NULL. For general information about system functions, see “String functions” on page 62.
- The datatypes of the expressions must convert implicitly, or you must use the convert function.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute isnull.

See also

Function – convert

is_sec_service_on

Description

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

Syntax

is_sec_service_on(*security_service_nm*)

Parameters

security_service_nm
– is the name of the security service.

Examples

```
select is_sec_service_on("unifiedlogin")
```

Usage

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

```
select * from syssecmechs
```

The result might look something like:

```
sec_mech_name available_service
------------- -------------------
dce           unifiedlogin
dce           mutualauth
dce           delegation
dce           integrity
dce           confidentiality
dce           detectreplay
dce           detectseq
```

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

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute is_sec_service_on.

See also — *Function* – show_sec_services

lct_admin

Description — Manages the last-chance threshold.

Returns the current value of the last-chance threshold.

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

Syntax

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

Parameters

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

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

database_id
– specifies the database.

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

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

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

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

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

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

Examples

Example 1

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

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

Example 2

```
select lct_admin("logfull", 6)
```

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

Example 3

```
select lct_admin("reserve", 64)

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

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

Example 4

```
select lct_admin("reserve",0)
```

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

Example 5

```
select lct_admin("abort", 83)
```

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

Example 6

```
select lct_admin("abort", 0, 5)
```

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

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

Usage

- lct_admin, a system function, manages the log segment's last-chance threshold. For general information about system functions, see "String functions" on page 62.
- If lct_admin("lastchance", *dbid*) returns zero, the log is not on a separate segment in this database, so no last-chance threshold exists.
- Whenever you create a database with a separate log segment, the server creates a default last chance threshold that defaults to calling sp_thresholdaction. This happens even if a procedure called sp_thresholdaction does not exist on the server at all.

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

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

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

See also

Command – dump transaction

Function – curunreservedpgs

license_enabled

Description

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

Syntax

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

Parameters

ase_server
– specifies the license for Adaptive Server.

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

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

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

ase_asm
– specifies the license for Adaptive Server advanced security mechanism.

Examples

```
select license_enabled("ase_dtm")

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

Indicates that the license for the Adaptive Server distributed transaction management feature is enabled.

Usage

- For information about installing license keys for Adaptive Server features, see your *Installation Guide*.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute license_enabled.

See also

System procedure – sp_configure

lockscheme

Description

Returns the locking scheme of the specified object as a string.

Syntax

lockscheme(*object_name*)
or,
lockscheme(*dbid*, *object_id*)

Parameters

oject_name
– is the name of the object you are searching. If you do not specify a fully qualified object name, the current database is searched.

dbid
the ID of the database specified by *object_name*.

object_id
the ID of the object indicated by *object_name*

Examples

Example 1

```
select lockscheme(title)
from titles
```

Selects the locking scheme for the title column of the titles table.

Example 2

```
select lockscheme(4, 224000798)
```

Selects the locking scheme for *object_id* 224000798 (in this case, the titles table) from database ID 4 (the pubs2 database).

Usage

- lockscheme() returns varchar(11) and allows NULLs
- If the specified object is not a table, lockscheme() returns the string "not a table."
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute lockscheme().

log

Description

Returns the natural logarithm of the specified number.

Syntax

log(*approx_numeric*)

Parameters

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

Examples

```
select log(20)

--------------------
           2.995732
```

Usage

- log, a mathematical function, returns the natural logarithm of the specified value.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute log.

See also

Functions – log10, power

log10

Description

Returns the base 10 logarithm of the specified number.

Syntax

log10(*approx_numeric*)

Parameters

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

Examples

```
select log10(20)

--------------------
         1.301030
```

Usage

- log10, a mathematical function, returns the base 10 logarithm of the specified value.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute log10.

See also

Functions – log, power

lower

Description
Returns the lowercase equivalent of the specified expression.

Syntax
lower(*char_expr*|*uchar_expr*)

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

uchar_expr
– is a character-type column name, variable, or constant expression of unichar or univarchar type

Examples

```
select lower(city) from publishers

--------------------
boston
washington
berkeley
```

Usage

- lower, a string function, converts uppercase to lowercase, returning a character value.
- lower is the inverse of upper.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- For general information about string functions, see "String functions" on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute lower.

See also
Functions – upper

ltrim

Description
Returns the specified expression, trimmed of leading blanks.

Syntax
ltrim(char_expr|uchar_expr)

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

uchar_expr
–is a character-type column name, variable, or constant expression of unichar, or univarchar type.

Examples

```
select ltrim("    123")

-------
123
```

Usage

- ltrim, a string function, removes leading blanks from the character expression. Only values equivalent to the space character in the current character set are removed.
- If *char_expr or uchar_expr* is NULL, returns NULL.
- For Unicode expressions, returns the lower-case Unicode equivalent of the specified expression. Characters in the expression that have no lower-case equivalent are left unmodified.
- For general information about string functions, see "String functions" on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute ltrim.

See also
Functions – rtrim

max

Description
Returns the highest value in an expression.

Syntax
max(*expression*)

Parameters
expression
– is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery.

Examples

Example 1

```
select max(discount) from salesdetail

--------------------
            62.200000
```

Returns the maximum value in the discount column of the salesdetail table as a new column.

Example 2

```
select discount from salesdetail
compute max(discount)
```

Returns the maximum value in the discount column of the salesdetail table as a new row.

Usage

- max, an aggregate function, finds the maximum value in a column or expression. For general information about aggregate functions, see “Aggregate functions” on page 45.
- max can be used with exact and approximate numeric, character, and datetime columns. It cannot be used with bit columns. With character columns, max finds the highest value in the collating sequence. max ignores null values. max implicitly converts char datatypes to varchar, unichar datatypes to univarchar, stripping all trailing blanks.
- unichar data is collated according to the default Unicode sort order.
- Adaptive Server goes directly to the end of the index to find the last row for max when there is an index on the aggregated column, unless:
    - The *expression* not a column
    - The column is not the first column of an index
    - There is another aggregate in the query
    - There is a group by or where clause

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute max.

See also

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

Functions – avg, min

min

Description

Returns the lowest value in a column.

Syntax

min(*expression*)

Parameters

expression
– is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" on page 179.

Examples

```
select min(price) from titles
where type = "psychology"

------------------------
                    7.00
```

Usage

- min, an aggregate function, finds the minimum value in a column.
- For general information about aggregate functions, see "Aggregate functions" on page 45.
- min can be used with numeric, character, and datetime columns. It cannot be used with bit columns. With character columns, min finds the lowest value in the sort sequence. min implicitly converts char datatypes to varchar, unichar datatypes to univarchar, stripping all trailing blanks. min ignores null values. distinct is not available, since it is not meaningful with min.
- unichar data is collated according to the default Unicode sort order.
- Adaptive Server goes directly to the first qualifying row for min when there is an index on the aggregated column, unless:
  - The *expression* is not a column
  - The column is not the first column of an index
  - There is another aggregate in the query
  - There is a group by clause

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute min.

See also

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

Functions – avg, max

mut_excl_roles

Description

Returns information about the mutual exclusivity between two roles.

Syntax

mut_excl_roles (*role1, role2* [membership | activation])

Parameters

role1
– is one user-defined role in a mutually exclusive relationship.

role2
– is the other user-defined role in a mutually exclusive relationship.

level
– is the level (membership or activation) at which the specified roles are exclusive.

Examples

```
alter role admin add exclusive membership supervisor
select
mut_excl_roles("admin", "supervisor", "membership")

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

Shows that the admin and supervisor roles are mutually exclusive.

Usage

- mut_excl_roles, a system function, returns information about the mutual exclusivity between two roles. If the System Security Officer defines role1 as mutually exclusive with role2 or a role directly contained by role2, mut_excl_roles returns 1. If the roles are not mutually exclusive, mut_excl_roles returns 0.
- For general information about system functions, see “System functions” on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute mut_excl_roles.

See also

Commands – alter role, create role, drop role, grant, set, revoke

Functions – proc_role, role_contain, role_id, role_name

System procedures – sp_activeroles, sp_displayroles, sp_role

CHAPTER 5

Functions: *object_id – rtrim*

object_id

Description

Returns the object ID of the specified object.

Syntax

object_id(*object_name*)

Parameters

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

Examples

Example 1

```
select object_id("titles")

-----------
  208003772
```

Example 2

```
select object_id("master..sysobjects")

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

Usage

- object_id, a system function, returns the object's ID. Object IDs are stored in the id column of sysobjects.
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute object_id.

See also

Functions – col_name, db_id, object_name

System procedure – sp_help

object_name

Description

Returns the name of the object whose object ID is specified.

Syntax

object_name(*object_id*[, *database_id*])

Parameters

object_id
– is the object ID of a database object, such as a table, view, procedure, trigger, default, or rule. Object IDs are stored in the id column of sysobjects.

database_id
– is the ID for a database if the object is not in the current database. Database IDs* are stored in the db_id column of sysdatabases.

Examples

Example 1

```
select object_name(208003772)

------------------------------
titles
```

Example 2

```
select object_name(1, 1)

------------------------------
sysobjects
```

Usage

- object_name, a system function, returns the object's name.
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute object_name.

See also

Functions – col_name, db_id, object_id

System procedures – sp_help

patindex

Description

Returns the starting position of the first occurrence of a specified pattern.

Syntax

patindex("%*pattern*%", *char_expr*|*uchar_expr* [, using
    {bytes | characters | chars}])

Parameters

pattern
– is a character expression of the char or varchar datatype that may include any of the pattern-match wildcard characters supported by Adaptive Server. The % wildcard character must precede and follow *pattern* (except when searching for first or last characters). For a description of the wildcard characters that can be used in *pattern*, see “Pattern matching with wildcard characters” on page 195.

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

uchar_expr
– is a character-type column name, variable, or constant expression of unichar, or univarchar type.

using
– specifies a format for the starting position.

bytes
– returns the offset in bytes.

chars
or characters – returns the offset in characters (the default).

Examples

Example 1

```
select au_id, patindex("%circus%", copy)
from blurbs
```

```
au_id
 ----------- -----------
 486-29-1786           0
 648-92-1872           0
 998-72-3567          38
 899-46-2035          31
 672-71-3249           0
 409-56-7008           0
```

Selects the author ID and the starting character position of the word “circus” in the copy column.

Example 2

```
select au_id, patindex("%circus%", copy,
     using chars)
from blurbs
```

Example 3

```
select au_id, patindex("%circus%", copy,
     using chars)
from blurbs
```

The same as example 1.

Example 4

```
select name
from sysobjects
where patindex("sys[a-d]%", name) > 0
```

```
name
------------------------------
sysalternates
sysattributes
syscharsets
syscolumns
syscomments
sysconfigures
sysconstraints
syscurconfigs
sysdatabases
sysdepends
sysdevices
```

Finds all the rows in sysobjects that start with “sys” and whose fourth character is “a”, “b”, “c”, or “d”.

Usage

- patindex, a string function, returns an integer representing the starting position of the first occurrence of *pattern* in the specified character expression, or a zero if *pattern* is not found.
- patindex can be used on all character data, including text and image data.
- By default, patindex returns the offset in characters; to return the offset in bytes (multibyte character strings), specify using bytes.
- Include percent signs before and after *pattern*. To look for *pattern* as the first characters in a column, omit the preceding %. To look for *pattern* as the last characters in a column, omit the trailing %.
- If *char_expr* or *uchar_expr* is NULL, returns 0.
- If a varchar expression is given as one parameter and a unichar expression is given as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

- For general information about string functions, see “String functions” on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute patindex.

See also
Functions – charindex, substring

pagesize

Description
Returns the page size, in bytes, for the specified object.

Syntax
pagesize(*object_name* [, *index_name*)
or,
pagesize(dbid, object_id [, index_id])

Parameters
object_name
- the name of the object you are searching. If you do not specify a fully qualified object name, the current database is searched.

index_name
– indicates the name of the index used for the search

dbid
– the ID of the database specified by *object_name*.

object_id
– the ID of the object indicated by *object_name* .

index_id
- the ID of the index indicated by *index_name*.

Examples
Example 1

```
select pagesize(title, title_id)
from titles
```

Selects the pagesize for the title column of the titles table.

Example 2

```
select pagesize(4, )
```

Selects the pagesize for the titles table (object_id 224000798) from the pubs2 database (dbid 4).

Usage
- If you do not indicate an *index_name*, the default is to use the data level of the table.

- If the specified object is not a page size (for example, if the name of a view is provided), pagesize() returns zero.
- If the specified object does not exist, pagesize() returns NULL.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute pagesize().

pi

Description — Returns the constant value 3.1415926535897936.

Syntax — pi()

Parameters — None

Examples

```
select pi()

-------------------
          3.141593
```

Usage

- pi, a mathematical function, returns the constant value of 3.1415926535897931.
- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute pi.

See also — *Functions* – degrees, radians

power

Description — Returns the value that results from raising the specified number to a given power.

Syntax — power(*value*, *power*)

Parameters — *value*
 – is a numeric value.

power
– is an exact numeric, approximate numeric, or money value.

Examples

```
select power(2, 3)

-----------
          8
```

Usage

- power, a mathematical function, returns the value of *value* raised to the power *power*. Results are of the same type as *value*.

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

- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute power.

See also

Functions – exp, log, log

proc_role

Description

Returns information about whether the user has been granted the specified role.

Syntax

proc_role ("*role_name*")

Parameters

role_name
– is the name of a system or user-defined role.

Examples

Example 1

```
create procedure sa_check as
if (proc_role("sa_role") > 0)
begin
     return(1)
end
print "You are a System Administrator."
```

Creates a procedure to check if the user is a System Administrator.

Example 2

```
select proc_role("sso_role")
```

Checks that the user has been granted the System Security Officer role.

Example 3

```
select proc_role("oper_role")
```

Checks that the user has been granted the Operator role.

Usage

- proc_role, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- proc_role returns 0 if any of the following are true:
  - the user has not been granted the specified role
  - the user has not been granted a role which contains the specified role
  - the user has been granted, but has not activated, the specified role
- proc_role returns 1 if the invoking user has been granted, and has activated, the specified role.
- proc_role returns 2 if the invoking user has a currently active role, which contains the specified role.
- For general information about system functions, see “System functions” on page 64.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute proc_role.

See also — *Commands* – alter role, create role, drop role, grant, set, revoke

Functions – mut_excl_roles, role_contain, role_id, role_name, show_role

ptn_data_pgs

Description — Returns the number of data pages used by a partition.

Syntax — ptn_data_pgs(*object_id*, *partition_id*)

Parameters

object_id
– is the object ID for a table, stored in the id column of sysobjects, sysindexes, and syspartitions.

partition_id
– is the partition number of a table.

Examples

```
select ptn_data_pgs(object_id("salesdetail"), 1)

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

Usage

- ptn_data_pgs, a system function, returns the number of data pages in a partitioned table.
- Use the object_id function to get an object's ID, and use sp_helpartiton to list the partitions in a table.
- The data pages returned by ptn_data_pgs may be inaccurate. Use the update partition statistics, dbcc checktable, dbcc checkdb, or dbcc checkalloc commands before using ptn_data_pgs to get the most accurate value.
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Only the table owner can execute ptn_data_pgs.

See also

Commands – update partition statistics, dbcc

Functions – data_pgs, object_id

System procedures – sp_helpartition

radians

Description

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

Syntax

radians(*numeric*)

Parameters

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

Examples

```
select radians(2578)

-----------
         44
```

Usage

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

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

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

- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute radians.

See also

Function – degrees

rand

Description

Returns a random value between 0 and 1, which is generated using the specified seed value.

Syntax

rand([*integer*])

Parameters

integer
– is any integer (tinyint, smallint or int) column name, variable, constant expression, or a combination of these.

Examples

Example 1

```
select rand()

--------------------
           0.395740
```

Example 2

```
declare @seed int
select @seed=100
select rand(@seed)

--------------------
             0.000783
```

Usage

- rand, a mathematical function, returns a random float value between 0 and 1, using the optional integer as a seed value.

- The rand function uses the output of a 32-bit pseudo-random integer generator. The integer is divided by the maximum 32-bit integer to give a double value between 0.0 and 1.0. The rand function is seeded randomly at server start-up, so getting the same sequence of random numbers is unlikely, unless the user first initializes this function with a constant seed value. The rand function is a global resource. Multiple users calling the rand function progress along a single stream of pseudo-random values. If a repeatable series of random numbers is needed, the user must assure that the function is seeded with the same value initially and that no other user calls rand while the repeatable sequence is desired.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute rand.

See also

Datatypes – Approximate numeric datatypes

replicate

Description

Returns a string consisting of the specified expression repeated a given number of times.

Syntax

replicate (*char_expr*|uchar_expr, *integer_expr*)

Parameters

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

uchar_expr
– is a character-type column name, variable, or constant expression of unichar or univarchar type.

integer_expr
– is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Examples

```
select replicate("abcd", 3)

------------
abcdabcdabcd
```

Usage

- replicate, a string function, returns a string with the same datatype as *char_expr*, or *uchar_expr* containing the same expression repeated the specified number of times or as many times as will fit into a 64K-space, whichever is less.
- If *char_expr* or *uchar_expr* is NULL, returns a single NULL.
- For general information about string functions, see "String functions" on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute replicate.

See also

Functions – stuff

reserved_pgs

Description

Returns the number of pages allocated to the specified table or index, and reports pages used for internal structures.

Syntax

reserved_pgs(*object_id,* {doampg|ioampg})

Parameters

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

doampg | ioampg
– specifies table (doampg) or index (ioampg).

Examples

```
select reserved_pgs(id, doampg)
from sysindexes where id =
    object_id("syslogs")

-------------
          534
```

Returns the page count for the syslogs table.

Usage

- reserved_pgs, a system function:
    - Returns the number of pages allocated to a table or an index
    - Reports pages used for internal structures
    - Works only on objects in the current database
- For general information about system functions, see "System functions" on page 64.

Standards	SQL92 – Complience level: Transact-SQL extension
Permissions	Any user can execute reserved_pgs.
See also	*Commands* – update statistics
	Functions – data_pgs

reverse

Description
Returns the specified string with characters listed in reverse order.

Syntax
reverse(*expression*|*uchar_expr*)

Parameters
expression
– is a character or binary-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, binary, or varbinary type.

uchar_expr
–is a character or binary-type column name, variable, or constant expression of unichar or univarchar type.

Examples
Example 1

```
select reverse("abcd")

----
dcba
```

Example 2

```
select reverse(0x12345000)

----------
0x00503412
```

Usage

- reverse, a string function, returns the reverse of *expression*.
- If *expression* is NULL, returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.
- For general information about string functions, see "String functions" on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute reverse.

See also
Functions – lower, upper

right

Description

The rightmost part of the expression with the specified number of characters.

Syntax

right(*expression*, *integer_expr*)

Parameters

expression

– is a character or binary-type column name, variable, or constant expression of char, varchar, nchar, unichar, nvarchar, univarchar, binary, or varbinary type.

integer_expr

– is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Examples

Example 1

```
select right("abcde", 3)

 ---
 cde
```

Example 2

```
select right("abcde", 2)

 --
 de
```

Example 3

```
select right("abcde", 6)

 -----
 abcde
```

Example 4

```
select right(0x12345000, 3)

 -------
 0x345000
```

Example 5

```
select right(0x12345000, 2)

 ------
 0x5000
```

Example 6

```
select right(0x12345000, 6)

 ---------
 0x12345000
```

Usage

- right, a string function, returns the specified number of characters from the rightmost part of the character or binary expression.
- If the specified rightmost part begins with the second surrogate of a pair (the low surrogate), the return value starts with the next full character. Therefore, one less character is returned.
- The return value has the same datatype as the character or binary expression.
- If *expression* is NULL, returns NULL.
- For general information about string functions, see "String functions" on page 62.

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute right.

See also *Functions* – rtrim, substring

role_contain

Description Returns 1 if *role2* contains *role1*.

Syntax role_contain("role1", "role2")

Parameters

role1
– is the name of a system or user-defined role.

role2
– is the name of another system or user-defined role.

Examples

Example 1

```
select role_contain("intern_role", "doctor_role")

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

Example 2

```
select role_contain("specialist_role",
"intern_role")
-----------
0
```

Usage

- role_contain, a system function, returns 1 if *role1* is contained by *role2*.
- For more information about contained roles and role hierarchies, see the *System Administration Guide*.
- For more information about system functions, see "System functions" on page 64

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute role_contain.

See also

Functions – mut_excl_roles, proc_role, role_id, role_name

Commands – alter role

System procedures – sp_activeroles, sp_displayroles, sp_role

role_id

Description

Returns the system role ID of the role whose name you specify.

Syntax

role_id("*role_name*")

Parameters

role_name
– is the name of a system or user-defined role. Role names and role IDs are stored in the syssrvroles system table.

Examples

Example 1

```
select role_id("sa_role")

------
0
```

Returns the system role ID of sa_role.

Example 2

```
select role_id("intern_role")
```

```
------
6
```

Returns the system role ID of the "intern_role".

Usage

- role_id, a system function, returns the system role ID (srid). System role IDs are stored in the srid column of the syssrvroles system table.
- If the *role_name* is not a valid role in the system, Adaptive Server returns NULL.
- For more information about roles, see the *System Administration Guide*.
- For more information about system functions, see "System functions" on page 64.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute role_id.

See also — *Functions* – mut_excl_roles, proc_role, role_contain, role_name

role_name

Description — Returns the name of a role whose system role ID you specify.

Syntax — role_name(*role_id*)

Parameters — *role_id*
 – is the system role ID (srid) of the role. Role names are stored in syssrvroles.

Examples

```
select role_name(01)

-----------------------------
sso_role
```

Usage

- role_name, a system function, returns the role name.
- For more information about system functions, see "System functions" on page 64.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute role_name.

See also — *Functions* – mut_excl_roles, proc_role, role_contain, role_id

round

Description

Returns the value of the specified number, rounded to a given number of decimal places.

Syntax

round(*number*, *decimal_places*)

Parameters

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

decimal_places
– is the number of decimal places to round to.

Examples

Example 1

```
select round(123.4545, 2)

----------
  123.4500
```

Example 2

```
select round(123.45, -2)

----------
  100.00
```

Example 3

```
select round(1.2345E2, 2)

-----------------
       123.450000
```

Example 4

```
select round(1.2345E2, -2)

-----------------
      100.000000
```

Usage

- round, a mathematical function, rounds the *number* so that it has *decimal_places* significant digits.
- A positive *decimal_places* determines the number of significant digits to the right of the decimal point; a negative *decimal_places*, the number of significant digits to the left of the decimal point.
- Results are of the same type as *number* and, for numeric and decimal expressions, have an internal precision equal to the precision of the first argument plus 1 and a scale equal to that of *number*.

- round always returns a value. If *decimal_places* is negative and exceeds the number of significant digits in *number*, Adaptive Server returns a result of 0. (This is expressed in the form 0.00, where the number of zeros to the right of the decimal point is equal to the scale of numeric.) For example:

  ```
  select round(55.55, -3)
  ```

  returns a value of 0.00.

- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute round.

See also

Functions – abs, ceiling, floor, sign, str

rowcnt

Description

Returns an estimate of the number of rows in the specified table.

Syntax

rowcnt(sysindexes.doampg)

Parameters

sysindexes.doampg
– is the row count maintained in sysindexes.

Examples

```
select name, rowcnt(sysindexes.doampg)
     from sysindexes
     where name in
            (select name from sysobjects
             where type = "U")

name
----------------------------- ----------
roysched                              87
salesdetail                          116
stores                                 7
discounts                              4
au_pix                                 0
blurbs                                 6
```

Usage

- rowcnt, a system function, returns the estimated number of rows in a table.

- The value returned by rowcnt can vary unexpectedly when Adaptive Server reboots and recovers transactions. The value is most accurate after running one of the following commands:
  - dbcc checkalloc
  - dbcc checkdb
  - dbcc checktable
  - update all statistics
  - update statistics
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute rowcnt.

See also

Catalog stored procedures – sp_statistics

Commands – dbcc, update all statistics, update statistics

Function – data_pgs

System procedures – sp_helpartition, sp_spaceused

rtrim

Description

Returns the specified expression, trimmed of trailing blanks.

Syntax

rtrim(*char_expr*|*uchar_expr*)

Parameters

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

uchar_expr
–is a character-type column name, variable, or constant expression of unichar, or univarchar type.

Examples

```
select rtrim("abcd    ")

--------
abcd
```

Usage

- rtrim, a string function, removes trailing blanks.

- For Unicode, a blank is defined as the Unicode value U+0020.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- Only values equivalent to the space character in the current character set are removed.
- For general information about string functions, see “String functions” on page 62.

Standards	SQL92 – Complience level: Transact-SQL extension
Permissions	Any user can execute rtrim.
See also	*Functions* – ltrim

CHAPTER 6

Functions: *show_role – valid_user*

show_role

Description	Shows the login's currently active system-defined roles.
Syntax	show_role()
Parameters	None.

Examples

Example 1

```
select show_role()
```

```
sa_role sso_role oper_role replication_role
```

Example 2

```
if charindex("sa_role", show_role()) >0
begin
     print "You have sa_role"
end
```

Usage

- show_role, a system function, returns the login's current active system-defined roles, if any (sa_role, sso_role, oper_role, or replication_role). If the login has no roles, show_role returns NULL.
- When a Database Owner invokes show_role after using setuser, show_role displays the active roles of the Database Owner, not the user impersonated with setuser.
- For general information about system functions, see "System functions" on page 64.

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute show_role.

See also

Commands – alter role, create role, drop role, grant, set, revoke

Functions – proc_role, role_contain

System procedures – sp_activeroles, sp_displayroles, sp_role

show_sec_services

Description

Lists the security services that are active for the session.

Syntax

show_sec_services()

Parameters

None.

Examples

```
select show_sec_services()
```

```
encryption, replay_detection
```

Shows that the user's current session is encrypting data and performing replay detection checks.

Usage

- Use show_sec_services to list the security services that are active during the session.
- If no security services are active, show_sec_services returns NULL.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute show_sec_services.

See also

Functions – is_sec_service_on

sign

Description

Returns the sign (+1 for positive, 0, or -1 for negative) of the specified value.

Syntax

sign(*numeric*)

Parameters

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

Examples

Example 1

```
select sign(-123)
```

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

Example 2

```
select sign(0)
```

```
-----------
          0
```

Example 3

```
select sign(123)
```

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

Usage

- sign, a mathematical function, returns the positive (+1), zero (0), or negative (-1).
- Results are of the same type, and have the same precision and scale, as the numeric expression.
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute sign.

See also
Functions – abs, ceiling, floor, round

sin

Description
Returns the sine of the specified angle (in radians).

Syntax
sin(*approx_numeric*)

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

Examples

```
select sin(45)
```

```
--------------------
          0.850904
```

Usage

- sin, a mathematical function, returns the sine of the specified angle (measured in radians).
- For general information about mathematical functions, see "Mathematical functions" on page 60.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute sin.

See also *Functions* – cos, degrees, radians

sortkey

Description Generates values that can be used to order results based on collation behavior, which allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity.

Syntax sortkey (*char_expression*|uchar_expression) [, {*collation_name* | *collation_ID*}])

Parameters *char_expression*

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

uchar_expression

– is a character-type column name, variable, or constant expression of unichar or univarchar type.

collation_name

– is a quoted string or a character variable that specifies the collation to use. Table 6-1 shows the valid values.

collation_ID

is an integer constant or a variable that specifies the collation to use. Table 6-1 shows the valid values.

Examples **Example 1**

```
select * from cust_table where cust_name like “TI%” order by
(sortkey(cust_name, “dict”)
```

Shows sorting by European language dicitionary order.

Example 2

```
select *from cust_table where cust name like “TI%” order by (sortkey(cust-
name, “gbpinyin”)
```

Shows sorting by simplified Chinese phonetic order.

Example 3

Shows sorting by European language dictionary order using the in-line option.

```
select *from cust_table where cust_name like”TI%” order by cust_french_sort
```

Example 4

```
select * from cust_table where cust_name like “TI%” order by
cust_chinese_sort.
```

Shows sorting by Simplified Chinese phonetic order using pre-existing keys.

Usage

- sortkey, a system function, generates values that can be used to order results based on collation behavior. This allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity. The return value is a varbinary datatype value that contains coded collation information for the input string that is returned from the sortkey function.

  For example, you can store the values returned by sortkey in a column with the source character string. When you want to retrieve the character data in the desired order, the select statement only needs to include an order by clause on the columns that contain the results of running sortkey.

  sortkey guarantees that the values it returns for a given set of collation criteria work for the binary comparisons that are performed on varbinary datatypes.

  Note sortkey can generate up to 6 bytes of collation information for each input character. Therefore, the result from using sortkey may exceed the 255-byte length limit of the varbinary datatype. If this happens, the result is truncated to fit. Truncation removes result bytes for each input character until the result string is less than 255 bytes. If this occurs, a warning message is issued, but the query or transaction that contained the sortkey function continues to work.

- *char_expression* or *uchar_expression* must be composed of characters that are encoded in the server’s default character set.
- *char_expression* or *uchar_expression* can be an empty string. If it is an empty string:
  - sortkey returns a zero-length varbinary value, and
  - stores a blank for the empty string.

  An empty string has a different collation value than an NULL string from a database column.

- If *char_expression* or *uchar_expression* is NULL, sortkey returns a NULL value.
- If a unicode expression has no specified sort order, the unicode default sort order is used.
- If you do not specify a value for *collation_name* or *collation_ID*, sortkey assumes binary collation.

Collation Tables

There are two types of collation tables you can use to perform multilingual sorting:

1. A "built-in" collation table created by the sortkey function. This function exists in all ASE releases after ASE 11.5.1. You can use either the collation name or the collation ID to specify a built-in table.
2. An external collation table that uses the Unilib library sorting functions. You must use the collation name to specify an external table. These*srt files are located at $SYBASE/collate/unicode.

   Both of these methods work equally well, but a "built-in" table is tied to a Sybase ASE database, an external table is not. If you use an ASE database, a built-in table provides the best performance. both of these methods can handle any mix of English, European, and Asian languages.

There are two ways of using sortkey:

1. In-line: This uses sortkey as part of the order by clause and is useful for retrofitting an existing application and minimizing the changes. Note however, that this method generates sort keys on-the-fly, and therefore does not provide optimum performance on large datasets of over 1000 records.
2. Pre-existing keys: this method calls sortkey whenever a new record requiring multilingual sorting is added to the table, such as a new customer name. Shadow columns (binary or varbinary type) must be set up in the database, preferably in the same table, one for each desired sort order (e.g. French, Chinese, etc.). When a query requires output to be sorted, the order by clause uses one of the shadow columns. This method produces the best performance since keys are already generated and stored, and are quickly compared only on the basis of their binary values.

You can view a list of available collation rules. Print out the list by executing either the stored procedure sp_helpsort, or by querying and selecting the name, id, and description from syscharsets, (type is between 2003 and 2999.)

- Table 6-1 lists the valid values for *collation_name* and *collation_ID*.

Table 6-1: Collation names and IDs

Description	Collation name	Collation ID
Binary sort	binary	50
Default Unicode multilingual	default	0
CP 850 Alternative no accent	altnoacc	39
CP 850 Alternative lower case first	altdict	45
CP 850 Alternative no case preference	altnocsp	46
CP 850 Scandinavian dictionary	scandict	47
CP 850 Scandinavian no case preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52
Latin-1 English, French, German no case preference	nocasep	53
Latin-1 English, French, German no accent	noaccent	54
Latin-1 Spanish dictionary	espdict	55
Latin-1 Spanish no case	espnocs	56
Latin-1 Spanish no accent	espnoac	57
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Russian dictionary	rusdict	58
ISO 8859-9 Turkish dictionary	turdict	72
Shift-JIS binary order	sjisbin	259
Thai dictionary	thaidict	1

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute sortkey.

See also *Functions* – compare

soundex

Description

Returns a 4-character code representing the way an expression sounds.

Syntax

soundex(*char_expr*|*uchar_expr*)

Parameters

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

uchar_expr
–is a character-type column name, variable, or constant expression of unichar or univarchar type.

Examples

```
select soundex ("smith"), soundex ("smythe")

----- -----
S530  S530
```

Usage

- soundex, a string function, returns a 4-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte roman letters.
- The soundex function converts an alpha string to a four-digit code for use in locating similar-sounding words or names. All vowels are ignored unless they constitute the first letter of the string.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute soundex.

See also

Functions – difference

space

Description

Returns a string consisting of the specified number of single-byte spaces.

Syntax

space(*integer_expr*)

Parameters

integer_expr
– is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Examples

```
select "aaa", space(4), "bbb"

--- ---- ---
aaa      bbb
```

Usage

- space, a string function, returns a string with the indicated number of single-byte spaces.
- For general information about string functions, see “String functions” on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute space.

See also

Functions – isnull, rtrim

sqrt

Description

Returns the square root of the specified number.

Syntax

sqrt(*approx_numeric*)

Parameters

approx_numeric
– is any approximate numeric (float, real, or double precision) column name, variable, or constant expression that evaluates to a positive number.

Examples

```
select sqrt(4)

 2.000000
```

Usage

- sqrt, a mathematical function, returns the square root of the specified value.
- If you attempt to select the square root of a negative number, Adaptive Server returns the following error message:

  ```
  Domain error occurred.
  ```

- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute sqrt.

See also

Functions – power

str

Description

Returns the character equivalent of the specified number.

Syntax

str(*approx_numeric* [, *length* [, *decimal*]])

Parameters

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

length
– sets the number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.

decimal
– sets the number of decimal digits to be returned. The default is 0.

Examples

Example 1

```
select str(1234.7, 4)

----
1235
```

Example 2

```
select str(-12345, 6)

------
-12345
```

Example 3

```
select str(123.45, 5, 2)

-----
123.5
```

Usage

- str, a string function, returns a character representation of the floating point number. For general information about string functions, see "String functions" on page 62.
- *length* and *decimal* are optional. If given, they must be nonnegative. str rounds the decimal portion of the number so that the results fit within the specified length. The length should be long enough to accommodate the decimal point and, if negative, the number's sign. The decimal portion of the result is rounded to fit within the specified length. If the integer portion of the number does not fit within the length, however, str returns a row of asterisks of the specified length. For example:

```
select str(123.456, 2, 4)
--
**
```

A short *approx_numeric* is right justified in the specified length, and a long *approx_numeric* is truncated to the specified number of decimal places.

- If *approx_numeric* is NULL, returns NULL.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute str.

See also
Functions – abs, ceiling, floor, round, sign

stuff

Description
Returns the string formed by deleting a specified number of characters from one string and replacing them with another string.

Syntax
stuff(*char_expr1*|*uchar_expr1*, *start*, *length*, *char_expr2*|*uchar_expr2*)

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

uchar_expr1
–is a character-type column name, variable, or constant expression of unichar or univarchar type.

start
– specifies the character position at which to begin deleting characters.

length
– specifies the number of characters to delete.

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

uchar_expr2
– is another character-type column name, variable, or constant expression of unichar or univarchar type.

Examples

Example 1

```
select stuff("abc", 2, 3, "xyz")

----
axyz
```

Example 2

```
select stuff("abcdef", 2, 3, null)

go
---
aef
```

Example 3

```
select stuff("abcdef", 2, 3, "")

----
a ef
```

Usage

- stuff, a string function, deletes *length* characters from *char_expr1* or *uchar_expr1* at *start*, then inserts *char_expr2 or uchar_expr2* into *char_expr1* or *uchar_expr2* at *start*. For general information about string functions, see "String functions" on page 62.
- If the start position or the length is negative, a NULL string is returned. If the start position is longer than *expr1*, a NULL string is returned. If the length to be deleted is longer than *expr1*, *expr1* is deleted through its last character (see example 1).
- If the start position falls in the middle of a surrogate pair, start is adjusted to be one less. If the start length position falls in the middle of a surrogate pair, length is adjusted to be one less.
- To use stuff to delete a character, replace *expr2* with "NULL" rather than with empty quotation marks. Using ' ' to specify a null character replaces it with a space (see examples 2 and 3).
- If *char_expr1* or *uchar_expr1* is NULL, returns NULL. If *char_expr1 or uchar_expr1* is a string value and *char_expr2* or *uchar_expr2* is NULL, replaces the deleted characters with nothing.
- If a varchar expression is given as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute stuff.

See also

Functions – replicate, substring

substring

Description

Returns the string formed by extracting the specified number of characters from another string.

Syntax

substring(*expression*, *start, length*)

Parameters

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

start
– specifies the character position at which the substring begins.

length
– specifies the number of characters in the substring.

Examples

Example 1

```
select au_lname, substring(au_fname, 1, 1)
from authors
```

Displays the last name and first initial of each author, for example, “Bennet A.”

Example 2

```
select substring(upper(au_lname), 1, 3)
from authors
```

Converts the author’s last name to uppercase, then displays the first three characters.

Example 3

```
select substring((pub_id + title_id), 1, 6)
from titles
```

Concatenates pub_id and title_id, then displays the first six characters of the resulting string.

Example 4

```
select substring(xactid,5,2)
```

```
from syslogs
```

Extracts the lower four digits from a binary field, where each position represents two binary digits.

Usage

- substring, a string function, returns part of a character or binary string. For general information about string functions, see "String functions" on page 62.
- If any of the arguments to substring are NULL, substring returns NULL.
- If the start position from the beginning of uchar_expr1 falls in the middle of a surrogate pair, start is adjusted to one less. If the start length position from the beginning of uchar_expr1 falls in the middle of a surrogate pair, length is adjusted to one less.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute substring.

See also

Functions – charindex, patindex, stuff

sum

Description

Returns the total of the values.

Syntax

sum([all | distinct] *expression*)

Parameters

all
– applies sum to all values. all is the default.

distinct
– eliminates duplicate values before sum is applied. distinct is optional.

expression
– is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see "Expressions" on page 179.

Examples

Example 1

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

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

Example 2

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

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

Example 3

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

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

Usage

- sum, an aggregate function, finds the sum of all the values in a column. sum can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating sums.
- For general information about aggregate functions, see "Aggregate functions" on page 45.
- When you sum integer data, Adaptive Server treats the result as an int value, even if the datatype of the column is smallint or tinyint. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type int.
- You cannot use sum with the binary datatypes.
- Since this function only defines numeric types, use with Unicode expressions generates an error.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute sum.

See also

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

Functions – count, max, min

suser_id

Description
Returns the server user's ID number from the syslogins table.

Syntax
suser_id([*server_user_name]*)

Parameters
server_user_name
– is an Adaptive Server login name.

Examples
Example 1

```
select suser_id()

------
     1
```

Example 2

```
select suser_id("margaret")

------
     5
```

Usage

- suser_id, a system function, returns the server user's ID number from syslogins. For general information about system functions, see "System functions" on page 64.
- To find the user's ID in a specific database from the sysusers table, use the user_id system function.
- If no *server_user_name* is supplied, suser_id returns the server ID of the current user.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute suser_id.

See also
Functions – suser_name, user_id

suser_name

Description
Returns the name of the current server user or the user whose server ID is specified.

Syntax
suser_name([*server_user_id*])

Parameters
server_user_name
– is an Adaptive Server user ID.

Examples

Example 1

```
select suser_name()

------------------------------
sa
```

Example 2

```
select suser_name(4)

------------------------------
margaret
```

Usage

- suser_name, a system function, returns the server user's name. Server user IDs are stored in syslogins. If no *server_user_id* is supplied, suser_name returns the name of the current user.
- For general information about system functions, see "System functions" on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute suser_name.

See also

Functions – suser_id, user_name

syb_sendmsg

Description

Sends a message to a User Datagram Protocol (UDP) port.

Syntax

syb_sendmsg *ip_address*, *port_number*, *message*

Parameters

ip_address
– is the IP address of the machine where the UDP application is running.

port_number
– is the port number of the UDP port.

message
– is the message to send. It can be up to 255 characters in length.

Examples

Example 1

```
select syb_sendmsg("120.10.20.5", 3456, "Hello")
```

Sends the message "Hello" to port 3456 at IP address 120.10.20.5.

Example 2

```
declare @msg varchar(255)
select @msg = "Message to send"
select syb_sendmsg (ip_address, portnum, @msg)
from sendports
where username = user_name()
```

Reads the IP address and port number from a user table, and uses a variable for the message to be sent.

Usage

- syb_sendmsg is not supported on Windows NT.
- To enable the use of UDP messaging, a System Security Officer must set the configuration parameter allow sendmsg to 1.
- No security checks are performed with syb_sendmsg. Sybase strongly recommends caution when using syb_sendmsg to send sensitive information across the network. By enabling this functionality, the user accepts any security problems which result from its use.
- For a sample C program that creates a UDP port, see sp_sendmsg.

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute syb_sendmsg.

See also *System procedure* – sp_sendmsg

tan

Description Returns the tangent of the specified angle (in radians).

Syntax tan(*angle*)

Parameters *angle*

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

Examples

```
select tan(60)

-------------------
           0.320040
```

Usage

- tan, a mathematical function, returns the tangent of the specified angle (measured in radians).

- For general information about mathematical functions, see “Mathematical functions” on page 60.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute tan.

See also

Functions – atan, atn2, degrees, radians

textptr

Description

Returns a pointer to the first page of a text or image column.

Syntax

textptr(*column_name*)

Parameters

column_name
– is the name of a text column.

Examples

Example 1

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "486-29-1786"
readtext blurbs.copy @val 1 5
```

This example uses the textptr function to locate the text column, *copy*, associated with *au_id* 486-29-1786 in the author’s *blurbs* table. The text pointer is put into a local variable *@val* and supplied as a parameter to the readtext command, which returns 5 bytes, starting at the second byte (offset of 1).

Example 2

```
select au_id, textptr(copy) from blurbs
```

Selects the *title_id* column and the 16-byte text pointer of the *copy* column from the *blurbs* table.

Usage

- textptr, a text and image function, returns the text pointer value, a 16-byte varbinary value.
- If a text or an image column has not been initialized by a non-null insert or by any update statement, textptr returns a NULL pointer. Use textvalid to check whether a text pointer exists. You cannot use writetext or readtext without a valid text pointer.

- For general information about text and image functions, see “Text and image functions” on page 65.

Note Trailing f in varbinary values are truncated when the values are stored in tables. If you are storing text pointer values in a table, use binary as the datatype for the column.

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute textptr.

See also: *Datatypes* – text and image datatypes

Functions – textvalid

textvalid

Description: Returns 1 if the pointer to the specified text column is valid; 0 if it is not.

Syntax: textvalid("*table_name.column_name*", *textpointer*)

Parameters: “*table_name.column_name*”
– is the name of a table and its text column.

textpointer
– is a text pointer value.

Examples:

```
select textvalid ("texttest.blurb", textptr(blurb))
from texttest
```

Reports whether a valid text pointer exists for each value in the blurb column of the texttest table.

Usage:

- textvalid, a text and image function, checks that a given text pointer is valid. Returns 1 if the pointer is valid or 0 if it is not.
- The identifier for a text or an image column must include the table name.
- For general information about text and image functions, see “Text and image functions” on page 65.

Standards: SQL92 – Complience level: Transact-SQL extension

Permissions: Any user can execute textvalid.

See also

Datatypes – text and image datatypes

Functions – textptr

to_unichar

Description

Returns a unichar expression having the value of the integer expression.

Syntax

to_unichar (integer_expr)

Parameters

integer_expr
– is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Usage

- to_unichar, a string function, converts a Unicode integer value to a Unicode character value.
- If a unichar expression refers to only half of a surrogate pair, an error message appears and the operation is aborted.
- If a *integer_expr* is NULL, returns NULL.
- For general information about string functions, see "String functions" on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute to_unichar.

See also

Datatypes – text and image datatypes

Functions – char

tsequal

Description

Compares timestamp values to prevent update on a row that has been modified since it was selected for browsing.

Syntax

tsequal(*browsed_row_timestamp*, *stored_row_timestamp*)

Parameters

browsed_row_timestamp
– is the timestamp column of the browsed row.

stored_row_timestamp
– is the timestamp column of the stored row.

Examples

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp, 0x0001000000002ea8)
```

Retrieves the timestamp column from the current version of the publishers table and compares it to the value in the timestamp column that has been saved. If the values in the two timestamp columns are equal, updates the row. If the values are not equal, returns an error message.

Usage

- tsequal, a system function, compares the timestamp column values to prevent an update on a row that has been modified since it was selected for browsing. For general information about system functions, see "System functions" on page 64.
- tsequal allows you to use browse mode without calling the dbqual function in DB-Library. Browse mode supports the ability to perform updates while viewing data. It is used in front-end applications using Open Client and a host programming language. A table can be browsed if its rows have been timestamped.
- To browse a table in a front-end application, append the for browse keywords to the end of the select statement sent to Adaptive Server. For example:

```
Start of select statement in an Open Client application
...
   for browse

Completion of the Open Client application routine
```

- The tsequal function should not be used in the where clause of a select statement, only in the where clause of insert and update statements where the rest of the where clause matches a single unique row.

  If a timestamp column is used as a search clause, it should be compared like a regular varbinary column; that is, timestamp1 = timestamp2.

Timestamping a new table for browsing

- When creating a new table for browsing, include a column named timestamp in the table definition. The column is automatically assigned a datatype of timestamp; you do not have to specify its datatype. For example:

```
create table newtable(col1 int, timestamp,
    col3 char(7))
```

Whenever you insert or update a row, Adaptive Server timestamps it by automatically assigning a unique varbinary value to the timestamp column.

Timestamping an existing table

- To prepare an existing table for browsing, add a column named timestamp with alter table. For example:

  ```
  alter table oldtable add timestamp
  ```

  adds a timestamp column with a NULL value to each existing row. To generate a timestamp, update each existing row without specifying new column values. For example:

  ```
  update oldtable
  set col1 = col1
  ```

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute tsequal.

See also — *Datatypes* – Timestamp datatype

uhighsurr

Description — Returns 1 if the Unicode value at position *start* is the high half of a surrogate pair (which should appear first in the pair). Returns 0 otherwise.

Syntax — uhighsurr(*uchar_expr*,start)

Parameters — *uchar_expr*
–is a character-type column name, variable, or constant expression of unichar, or univarchar type.

start
– specifies the character position to investigate.

Usage

- uhighsurr, a string function, allows you to write explicit code for surrogate handling. Specifically, if a substring starts on a Unicode character where uhighsurr() is true, you need to extract a substring of at least 2 Unicode values. (*substr* will not extract half of a surrogate pair.)
- If *uchar_expr* is NULL, returns NULL.

- For general information about string functions, see “String functions” on page 62.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute uhighsurr.

See also — *Functions* – ulowsurr

ulowsurr

Description — Returns 1 if the Unicode value at position *start* is the low half of a surrogate pair (which should appear second in the pair). Returns 0 otherwise.

Syntax — ulowsurr(*uchar_expr*,start)

Parameters — *uchar_expr*
–is a character-type column name, variable, or constant expression of unichar, or univarchar type.

start
– specifies the character position to investigate.

Usage

- ulowsurr, a string function, allows you to write explicit code around adjustments performed by substr(), stuff(), and right (). Specifically, if a substring ends on a Unicode value where ulowsurr() is true, the user knows to extract a substring of 1 less characters (or 1 more). substr () does not extract a string that contains an unmatched surrogate pair.
- If *uchar_expr* is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 62.

Standards — SQL92 – Complience level: Transact-SQL extension

Permissions — Any user can execute ulowsurr.

See also — *Functions* – uhighsurr

upper

Description

Returns the uppercase equivalent of the specified string.

Syntax

upper(*char_expr*)

Parameters

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

Examples

```
select upper("abcd")

----
ABCD
```

Usage

- upper, a string function, converts lowercase to uppercase, returning a character value.
- If *char_expr* or *uchar_expr* is NULL, returns NULL.
- Characters that have no upper-case equivalent are left unmodified.
- If a unichar expression is created containing only half of a surrogate pair, an error message appears and the operation is aborted.
- For general information about string functions, see “String functions” on page 62.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute upper.

See also

Functions – lower

uscalar

Description

Returns the Unicode scalar value for the first Unicode character in an expression..

Syntax

uscalar(u*char_expr*)

Parameters

uchar_expr
–is a character-type column name, variable, or constant expression of unichar, or univarchar type.

Examples

Usage

- uscalar, a string function, returns the Unicode value for the first Unicode character in an expression,.

- If *uchar_expr* is NULL, returns NULL.
- If uscalar is called on a *uchar_expr* containing an unmatched surrogate half, and error occurs and the operation is aborted.
- For general information about string functions, see "String functions" on page 62.

Standards
SQL92 – Complience level: Transact-SQL extension

Permissions
Any user can execute uscalar.

See also
Functions – ascii

used_pgs

Description
Returns the number of pages used by a table or index. For an all-pages-locked table with a clustered index, it returns the sum of the table and index pages.

Syntax
used_pgs(*object_id, doampg, ioampg*)

Parameters
object_id
– is the object ID of the table for which you want to see the used pages. To see the pages used by an index, specify the object ID of the table to which the index belongs.

doampg
– is the page number for the object allocation map of a table or clustered index, stored in the doampg column of sysindexes.

ioampg
– is the page number for the allocation map of a nonclustered index, stored in the ioampg column of sysindexes.

Examples
Example 1

```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("titles")

name          id          indid  doampg   ioampg
------------- ----------- ------ -------- -------
titleidind      208003772      1      560     552
titleind        208003772      2        0     456

select used_pgs(208003772, 560, 552)

-----------
```

```
          6
```

Returns the number of pages used by the data and clustered index of the titles table.

Example 2

```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("stores")

name          id          indid  doampg   ioampg
------------- ----------- ------ -------- -------
stores          240003886      0     464       0

select used_pgs(240003886, 464, 0)

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

Returns the number of pages used by the stores table, which has no index.

Usage

- used_pgs, a system function, returns:
    - For all-pages-locked tables with a clustered index, the sum of the table and index pages
    - For data-only-locked tables and tables with no clustered index, the number of used pages in the table
    - For clustered and nonclustered indexes on data-only-locked tables, the number of pages in the index
- In the examples, indid 0 indicates a table; indid 1 indicates a clustered index; an indid of 2–250 is a nonclustered index; and an indid of 255 is text or image data.
- used_pgs only works on objects in the current database.
- Each table and each index on a table has an object allocation map (OAM), which contains information about the number of pages allocated to and used by an object. This information is updated by most Adaptive Server processes when pages are allocated or deallocated. The sp_spaceused system procedure reads these values to provide quick space estimates. Some dbcc commands update these values while they perform consistency checks.
- For general information about system functions, see “System functions” on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute used_pgs.

See also

Functions – data_pgs, object_id

user

Description

Returns the name of the current user.

Syntax

user

Parameters

None.

Examples

```
select user
------
dbo
```

Usage

- user, a system function, returns the user’s name.
- If the sa_role is active, you are automatically the Database Owner in any database you are using. Inside a database, the user name of the Database Owner is always “dbo”.
- For general information about system functions, see “System functions” on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

Any user can execute user.

See also

Functions – user_name

user_id

Description

Returns the ID number of the specified user or of the current user in the database.

Syntax

user_id([*user_name*])

Parameters

user_name
– is the name of the user.

Examples

Example 1

```
select user_id()

------
     1
```

Example 2

```
select user_id("margaret")

------
     4
```

Usage

- user_id, a system function, returns the user's ID number. For general information about system functions, see "System functions" on page 64.
- user_id reports the number from sysusers in the current database. If no *user_name* is supplied, user_id returns the ID of the current user. To find the server user ID, which is the same number in every database on Adaptive Server, use suser_id.
- Inside a database, the "guest" user ID is always 2.
- Inside a database, the user_id of the Database Owner is always 1. If you have the sa_role active, you are automatically the Database Owner in any database you are using. To return to your actual user ID, use set sa_role off before executing user_id. If you are not a valid user in the database, Adaptive Server returns an error when you use set sa_role off.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

You must System Administrator or System Security Officer to use this function on a user_name other than your own.

See also

Commands – setuser

Functions – suser_id, user_name

user_name

Description

Returns the name within the database of the specified user or of the current user.

Syntax

user_name([*user_id*])

Parameters

user_id
– is the ID of a user.

Examples

Example 1

```
select user_name()

------------------------------
dbo
```

Example 2

```
select user_name(4)

------------------------------
margaret
```

Usage

- user_name, a system function, returns the user's name, based on the user's ID in the current database. For general information about system functions, see "System functions" on page 64.
- If no *user_id* is supplied, user_name returns the name of the current user.
- If the sa_role is active, you are automatically the Database Owner in any database you are using. Inside a database, the user_name of the Database Owner is always "dbo".

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

You must be a System Administrator or System Security Officer to use this function on a user_id other than your own.

See also

Functions – suser_name, user_id

valid_name

Description

Returns 0 if the specified string is not a valid identifier or a number other than 0 if the string is a valid identifier.

Syntax

valid_name(*character_expression*)

Parameters

character_expression
– is a character-type column name, variable, or constant expression of char, varchar, nchar or nvarchar type. Constant expressions must be enclosed in quotation marks.

Examples

```
create procedure chkname
```

```
@name varchar(30)
as
     if valid_name(@name) = 0
     print "name not valid"
```

Creates a procedure to verify that identifiers are valid.

Usage

- valid_name, a system function, returns 0 if the *character_ expression* is not a valid identifier (illegal characters, more than 30 bytes long, or a reserved word), or a number other than 0 if it is a valid identifier.
- Adaptive Server identifiers can be a maximum of 30 bytes in length, whether single-byte or multibyte characters are used. The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_) character. Temporary table names, which begin with the pound sign (#), and local variable names, which begin with the at sign (@), are exceptions to this rule. valid_name returns 0 for identifiers that begin with the pound sign (#) and the at sign (@).
- For general information about system functions, see “System functions” on page 64.

Standards SQL92 – Complience level: Transact-SQL extension

Permissions Any user can execute valid_name.

See also *System procedure* – sp_checkreswords

valid_user

Description Returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

Syntax valid_user(*server_user_id*)

Parameters

server_user_id
– is a server user ID. Server user IDs are stored in the suid column of syslogins.

Examples

```
select valid_user(4)

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

Usage

- valid_user, a system function, returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

- For general information about system functions, see “System functions” on page 64.

Standards

SQL92 – Complience level: Transact-SQL extension

Permissions

You must be a System Administrator or a System Security Officer to use this function on a server_user_id other than your own.

See also

System procedures – sp_addlogin, sp_adduser

CHAPTER 7

Expressions, Identifiers, and Wildcard Characters

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

Expressions

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

Table 7-1 lists the types of expressions that are used in Adaptive Server syntax statements.

Table 7-1: 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 7-2: Arithmetic operators

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

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

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

When you perform arithmetic operations on mixed datatypes, for example float and int, Adaptive Server follows specific rules for determining the type of the result. For more information, see Chapter 1, “System and User-Defined Datatypes.”

Bitwise operators

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

Table 7-3 summarizes the results for operands of 0 and 1. If either operand is NULL, the bitwise operator returns NULL:

Table 7-3: Truth tables for bitwise operations

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

| (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 7-4 use two tinyint arguments, A = 170 (10101010 in binary form) and B = 75 (01001011 in binary form).

Table 7-4: 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

String concatenation operator

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

```
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.”

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

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

Comparison operators

Adaptive Server uses the comparison operators listed in Table 7-5:

Table 7-5: Comparison operators

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

In comparing character data, < means closer to the beginning of the server's sort order and > means closer to the end of the sort order. Uppercase and lowercase letters are equal in a case-insensitive sort order. Use sp_helpsort to see the sort order for your Adaptive Server. Trailing blanks are ignored for comparison purposes. So, for example, "Dirk" is the same as "Dirk ".

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

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

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

Nonstandard operators

The following operators are Transact-SQL extensions:

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

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

any is used with <, >, or = and a subquery. It returns results when any value retrieved in the subquery matches the value in the where or having clause of the outer statement. For more information, see the Transact-SQL User's Guide.

all is used with < or > and a subquery. It returns results when all values retrieved in the subquery are less than (<) or greater than (>) the value in the where or having clause of the outer statement. For more information, see the Transact-SQL User's Guide.

in returns results when any value returned by the second expression matches the value in the first expression. The second expression must be a subquery or a list of values enclosed in parentheses. in is equivalent to = any.For more information, see where Clause.

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.

NULL 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 7-6 shows the results of logical operations, including those that involve null values:

Table 7-6: 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 185 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. If a unichar expression is compared to a char (varchar, nchar, nvarchar) expression, the latter is implicitly converted to unichar.

Using the empty string

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

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

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

Including quotation marks in character expressions

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

```
’I don’’t understand.’
```

With double quotes:

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

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

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

Using the continuation character

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

Identifiers

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

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

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

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

You cannot use a reserved word, such as a Transact-SQL command, as an identifier. For a complete list of reserved words, see Chapter 8, “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')
```

Identifying tables or columns by their qualified object name

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, you must redefine all 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 201).

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 199). 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 7-7 summarizes the wildcard characters:

Table 7-7: Wildcard characters used with like

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

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

The percent sign (%) wildcard character

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

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

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

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

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

The underscore (_) wildcard character

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

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

Bracketed ([ ]) characters

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

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

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

To find both "DeFrance" and "deFrance":

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

The caret (^) wildcard character

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

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

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

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

Using multibyte wildcard characters

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

Using wildcard characters as literal characters

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

Adaptive Server provides two types of escape characters:

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

Using square brackets ([ ])as escape characters

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

Table 7-8 shows examples of square brackets used as escape characters with like.

Table 7-8: Using square brackets to search for wildcard characters

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

Using the *escape* clause

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

Do not use existing wildcard characters as escape characters because:

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

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

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

If the pattern contains two literal occurrences of the character that happens to be the escape character, the string must contain four consecutive escape characters. If the escape character does not divide the pattern into pieces of one or two characters, Adaptive Server returns an error message. Table 7-9 shows examples of escape clauses used with like.

Table 7-9: Using the escape clause

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

Using wildcard characters with *datetime* data

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

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

```
where arrival_time = ’9:20’
```

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

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

CHAPTER 8 Reserved Words

Keywords, also known as reserved words, are words that have special meanings. This chapter lists Transact-SQL and SQL92 keywords.

Transact-SQL reserved words

The words in the following list are reserved by Adaptive Server as keywords (part of SQL command syntax). They cannot be used as names of database objects such as databases, tables, rules, or defaults. They can be used as names of local variables and as stored procedure parameter names.

To find the names of existing objects that are reserved words, use sp_checkreswords.

A

add, all, alter, and, any, arith_overflow, as, asc, at, authorization, avg

B

begin, between, break, browse, bulk, by

C

cascade, case, char_convert, check, checkpoint, close, clustered, coalesce, commit, compute, confirm, connect, constraint, continue, controlrow, convert, count, create, current, cursor

D

database, dbcc, deallocate, declare, default, delete, desc, deterministic, disk distinct, double, drop, dummy, dump

E

else, end, endtran, errlvl, errordata, errorexit, escape, except, exclusive, exec, execute, exists, exit, exp_row_size, external

F

fetch, fillfactor, for, foreign, from, func, function

G

goto, grant, group

H

having, holdlock

I

identity, identity_gap, identity_insert, identity_start, if, in, index, inout, insert, install, intersect, into, is, isolation

J

jar, join

K

key, kill

L

level, like, lineno, load, lock

M

max, max_rows_per_page, min, mirror, mirrorexit, modify

N

national, new, noholdlock, nonclustered, not, null, nullif, numeric_truncation

O

of, off, offsets, on, once, online, only, open, option, or, order, out, output, over

P

partition, perm, permanent, plan, precision, prepare, primary, print, privileges, proc, procedure, processexit, proxy_table, public

Q

quiesce

R

raiserror, read, readpast, readtext, reconfigure, references remove, reorg, replace, replication, reservepagegap, return, returns, revoke, role, rollback, rowcount, rows, rule

S

save, schema, select, set, setuser, shared, shutdown, some, statistics, stringsize, stripe, sum, syb_identity, syb_restree, syb_terminate

T

table, temp, temporary, textsize, to, tran, transaction, trigger, truncate, tsequal

U

union, unique, unpartition, update, use, user, user_option, using

V

values, varying, view

W

waitfor, when, where, while, with, work, writetext

SQL92 reserved words

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

The words in the following list are SQL92 keywords that are not reserved words in Transact-SQL.

A

absolute, action, allocate, are, assertion

B

bit, bit_length, both

C

cascaded, case, cast, catalog, char, char_length, character, character_length, coalesce, collate, collation, column, connection, constraints, corresponding, cross, current_date, current_time, current_timestamp, current_user

D

date, day, dec, decimal, deferrable, deferred, describe, descriptor, diagnostics, disconnect, domain

E

end-exec, exception, extract

F

false, first, float, found, full

G

get, global, go

H

hour

I

immediate, indicator, initially, inner, input, insensitive, int, integer, interval

J

join

L

language, last, leading, left, local, lower

M

match, minute, module, month

N

names, natural, nchar, next, no, nullif, numeric

O

octet_length, outer, output, overlaps

P

pad, partial, position, preserve, prior

R

real, relative, restrict, right

S

scroll, second, section, session_user , size , smallint, space, sql, sqlcode, sqlerror, sqlstate, substring, system_user

T

then, time, timestamp, timezone_hour, timezone_minute, trailing, translate, translation, trim, true

U

unknown, upper, usage

V

value, varchar

W

when, whenever, write, year

Z

zone

Potential SQL92 reserved words

If you are using the ISO/IEC 9075:1989 standard, also avoid using the words shown in the following list because these words may become SQL92 reserved words in the future.

A

after, alias, async

B

before, boolean, breadth

C

call, completion, cycle

D

data, depth, dictionary

E

each, elseif, equals

G

general

I

ignore

L

leave, less, limit, loop

M

modify

N

new, none

O

object, oid, old, operation, operators, others

P

parameters, pendant, preorder, private, protected

R

recursive, ref, referencing, resignal, return, returns, routine, row

S

savepoint, search, sensitive, sequence, signal, similar, sqlexception, structure

T

test, there, type

U

under

V

variable, virtual, visible

W

wait, without

CHAPTER 9 SQLSTATE Codes and Messages

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

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

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

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

Warnings

Adaptive Server currently detects only one SQLSTATE warning condition, which is described in Table 9-1:

Table 9-1: 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 9-2 through Table 9-8. 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 9-2: 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 9-3: 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, unichar, univarchar, 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 9-4: 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.
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.

Message	Value	Description
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 9-5: 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.
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.

Message	Value	Description
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 9-6: 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.
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.

Message	**Value**	**Description**
The size (*size*) given to the *object_name* exceeds the maximum. The largest size allowed is *size*.	42000	Occurs when: • The total size of all the columns in a table definition exceeds the maximum allowed row size. • The size of a single column or parameter exceeds the maximum allowed for its datatype.

Transaction rollbacks

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

Table 9-7: 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 9-8: 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