SYBASE®

Reference Manual: Building Blocks

Adaptive Server® Enterprise

15.5

DOCUMENT ID: DC36271-01-1550-01

LAST REVISED: November 2009

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

The *Adaptive Server Reference Manual* includes four guides to Sybase® Adaptive Server® Enterprise and the Transact-SQL® language:

- *Building Blocks* describes the "parts" of Transact-SQL: datatypes, built-in functions, global variables, expressions and identifiers, reserved words, and SQLSTATE errors. Before you can use Transact-SQL sucessfully, you must understand what these building blocks do and how they affect the results of Transact-SQL statements.
- *Commands* provides reference information about the Transact-SQL commands, which you use to create statements.
- *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.
- *Tables* provides reference information about the system tables, which store information about your server, databases, users, and other details of your server. It also 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," 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.
- Chapter 3, "Global Variables," lists the system-defined variables for Adaptive Server in a table that provides the name and a brief description of the returned status.
- Chapter 4, "Expressions, Identifiers, and Wildcard Characters," which provides information about using the Transact-SQL language.

- Chapter 5, "Reserved Words," provides information about the Transact-SQL and ANSI SQL keywords.
- Chapter 6, "SQLSTATE Codes and Messages," contains information about Adaptive Server SQLSTATE status codes and the associated messages.

Related documents

The Adaptive Server® Enterprise documentation set consists of:

- 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. To check for critical product or document information that was added after the release of the product CD, use the Sybase® Product Manuals Web site.
- The installation guide for your platform – describes installation, upgrading, and some configuration procedures for all Adaptive Server and related Sybase products.
- *New Feature Summary* – describes the new features in Adaptive Server, the system changes added to support those features, and changes that may affect your existing applications.
- *Active Messaging Users Guide* – describes how to use the Active Messaging feature to capture transactions (data changes) in an Adaptive Server Enterprise database, and deliver them as events to external applications in real time.
- *Component Integration Services Users Guide* – explains how to use Component Integration Services to connect remote Sybase and non-Sybase databases.
- The *Configuration Guide* for your platform – provides instructions for performing specific configuration tasks.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Historical Server Users Guide* – describes how to use Historical Server to obtain performance information from Adaptive Server.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.
- *Job Scheduler Users Guide* – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).

- *Migration Technology Guide* – describes strategies and tools for migrating to a different version of Adaptive Server.
- *Monitor Client Library Programmers Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.
- *Monitor Server Users Guide* – describes how to use Monitor Server to obtain performance statistics from Adaptive Server.
- *Monitoring Tables Diagram* – illustrates monitor tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.
- *Performance and Tuning Series* – is a series of books that explain how to tune Adaptive Server for maximum performance:
    - *Basics* – contains the basics for understanding and investigating performance questions in Adaptive Server.
    - *Improving Performance with Statistical Analysis* – describes how Adaptive Server stores and displays statistics, and how to use the set statistics command to analyze server statistics.
    - *Locking and Concurrency Control* – describes how to use locking schemes to improve performance, and how to select indexes to minimize concurrency.
    - *Monitoring Adaptive Server with sp_sysmon* – discusses how to use sp_sysmon to monitor performance.
    - *Monitoring Tables* – describes how to query Adaptive Server monitoring tables for statistical and diagnostic information.
    - *Physical Database Tuning* – describes how to manage physical data placement, space allocated for data, and the temporary databases.
    - *Query Processing and Abstract Plans* – explains how the optimizer processes queries, and how to use abstract plans to change some of the optimizer plans.
- *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 (regular size when viewed in PDF format).
- *Reference Manual* – is a series of books that contains detailed Transact-SQL® information:

- *Building Blocks* – discusses datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
- *Commands* – documents commands.
- *Procedures* – describes system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.
- *Tables* – discusses system tables, monitor tables, and dbcc tables.

- *System Administration Guide* –
  - *Volume 1* – provides an introduction to the basics of system administration, including a description of configuration parameters, resource issues, character sets, sort orders, and instructions for diagnosing system problems. The second part of *Volume 1* is an in-depth discussion about security administration.
  - *Volume 2* – includes instructions and guidelines for managing physical resources, mirroring devices, configuring memory and data caches, managing multiprocessor servers and user databases, mounting and unmounting databases, creating and using segments, using the reorg command, and checking database consistency. The second half of *Volume 2* describes how to back up and restore system and user databases.
- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.
- *Transact-SQL Users Guide* – documents Transact-SQL, the Sybase-enhanced version of the relational database language. This guide serves as a textbook for beginning users of the database management system, and also contains detailed descriptions of the pubs2 and pubs3 sample databases.
- *Troubleshooting: Error Messages Advanced Resolutions* – contains troubleshooting procedures for problems you may encounter. The problems discussed here are the ones the Sybase Technical Support staff hear about most often.
- *Encrypted Columns Users Guide* – describes how to configure and use encrypted columns with Adaptive Server.
- *In-Memory Database Users Guide* – describes how to configure and use in-memory databases.

- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.
- *Using Backup Server with IBM® Tivoli® Storage Manager* – describes how to set up and use the IBM Tivoli Storage Manager to create Adaptive Server backups.
- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Unified Agent and Agent Management Console* – describes the Unified Agent, which provides runtime services to manage, monitor, and control distributed Sybase resources.
- *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- *Web Services Users Guide* – explains how to configure, use, and troubleshoot Web services for Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.
- *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that are available in XML services.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Certification Report title to display the report.

❖ Finding the latest information on component certifications

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ Creating a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

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,procedure names, utility names, and other keywords display in sans serif font.	select sp_configure
Database names and datatypes are in sans serif font.	master database
Book names, file names, variables, and path names are in italics.	*System Administration Guide* *sql.ini* file *column_name* *$SYBASE/ASE* directory

Element	Example
Variables—or words that stand for values that you fill in—when they are part of a query or statement, are in italics in Courier font.	`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*]

  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 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*.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Adaptive Server HTML documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

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, which specify the type, size, and storage format of columns, stored procedure parameters, and local variables.

Datatype categories

Adaptive Server provides several system datatypes and the user-defined datatypes timestamp, sysname, and longsysname. 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 and longsysname datatypes	System tables
text, image, and unitext datatypes	Printable characters or hexadecimal-like data that requires more than the maximum column size provided by your server's logical page size.
Abstract datatypes	Adaptive Server supports abstract datatypes through Java classes. See *Java in Adaptive Server Enterprise* for more information.
User-defined datatypes	Defining objects that inherit the rules, default, null type, IDENTITY property, and base datatype of the datatypes listed in this table. text undergoes character-set conversion if client is using a different character set, image does not.

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: Adaptive Server system datatypes

Datatypes by category	Synonyms	Range	Bytes of storage
Exact numeric: integers			

Datatypes by category	Synonyms	Range	Bytes of storage
bigint		Whole numbers between 2^{63} and -2^{63} - 1 (from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807, inclusive.	8
int	integer	2^{31} -1 (2,147,483,647) to -2^{31} (-2,147,483,648	4
smallint		2^{15} -1 (32,767) to -2^{15} (-32,768)	2
tinyint		0 to 255 (Negative numbers are not permitted)	1
unsigned bigint		Whole numbers between 0 and 18,446,744,073,709,551,615	8
unsigned int		Whole numbers between 0 and 4,294,967,295	4
unsigned smallint		Whole numbers between 0 and 65535	2
Exact numeric: decimals			
numeric (p, s)		10^{38} -1 to -10^{38}	2 to 17
decimal (p, s)	dec	10^{38} -1 to -10^{38}	2 to 17
Approximate numeric			
float (precision)		machine dependent	4 for default precision < 16, 8 for default precision >= 16
double precision		machine dependent	8
real		machine dependent	4
Money			
smallmoney		214,748.3647 to -214,748.3648	4
money		922,337,203,685,477.5807 to -922,337,203,685,477.5808	8
Date/time			
smalldatetime		January 1, 1900 to June 6, 2079	4
datetime		January 1, 1753 to December 31, 9999	8
date		January 1, 0001 to December 31, 9999	4
time		12:00:00AM to 11:59:59:999PM	4
bigdatetime		January 1, 0001 to December 31, 9999 and 12:00.000000AM to 11:59:59.999999 PM	8

Datatypes by category	Synonyms	Range	Bytes of storage
bigtime		12:00:00.000000 AM to 11:59:59.999999 PM	8
Character			
char(n)	character	pagesize	n
varchar(n)	character varying, char varying	pagesize	actual entry length
unichar	Unicode character	pagesize	*n* * *@@unicharsize* (*@@unicharsize* equals 2)
univarchar	Unicode character varying, char varying	pagesize	actual number of characters * *@@unicharsize*
nchar(n)	national character, national char	pagesize	*n* * *@@ncharsize*
nvarchar(n)	nchar varying, national char varying, national character varying	pagesize	*@@ncharsize* * number of characters
text		2^{31} -1 (2,147,483,647) bytes or fewer	0 when uninitialized; multiple of 2K after initialization
unitext		1 – 1,073,741,823	0 when uninitialized; multiple of 2K after initialization
Binary			
binary(n)		pagesize	*n*
varbinary(n)		pagesize	actual entry length
image		2^{31} -1 (2,147,483,647) bytes or fewer	0 when uninitialized; multiple of 2K after initialization
Bit			
bit		0 or 1	1 (one byte holds up to 8 bit columns)

Datatypes of columns, variables, or parameters

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

To declare the datatype of a new column in a create table or alter table statement, use:

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)
```

You can also declare the datatype of a new column in a select into statement, use convert or cast:

```
select convert(double precision, x), cast ( int, y) into
    newtable from oldtable
```

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

To declare the datatype for a local variable in a batch or stored procedure, use:

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

Numeric literals

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.

Character literals

In versions of Adaptive Server earlier than 12.5.1, when the client's character set was different from the server's character set, conversions were generally enabled to allow the text of SQL queries to be converted to the server's character set before being processed. If any character could not be converted because it could not be represented in the server's character set, the entire query was rejected. This character set "bottleneck" has been removed as of Adaptive Server version 12.5.1.

You cannot declare the datatype of a character literal. Adaptive Server treats character literals as varchar, except those that contain characters that cannot be converted to the server's default character set. Such literals are treated as univarchar. This makes it possible to perform such queries as selecting unichar data in a server configured for "iso_1" using a "sjis" (Japanese) client. For example:

```
select * from mytable where unichar_column = '五'
```

Since the character literal cannot be represented using the char datatype (in "iso_1"), it is promoted to the unichar datatype, and the query succeeds.

Datatypes 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
smalldatet	13
intn	14
uintn	15
bigint	16
ubigint	17
int	18
uint	19
smallint	20
usmallint	21
tinyint	22
bit	23
univarchar	24
unichar	25
unitext	26
sysname	27
varchar	27
nvarchar	27
longsysnam	27
char	28
nchar	28
timestamp	29
varbinary	29
binary	30
text	31
image	32
date	33
time	34
daten	35
timen	36
bigdatetim	37
bigtime	38
bigdatetim	39
bigtimen	40
extended t	99

Note u<int type> is an internal representation. The correct syntax for unsigned types is unsigned {int | integer | bigint | smallint }

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 or has the least hierarchical value.

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 20; royalty is an int, which has a hierarchy of 18. 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)

Datatype conversions

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, hextoint, inttohex, hextobigint, and biginttohex functions. See “Datatype conversion functions” on page 63 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
date	daten
time	timen
float	floatn
bigint, int, smallint, and tinyint	intn
unsigned bigint, unsigned int, and unsigned smallint	uintn
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.

  Setting arith_overflow to on refers to the execution time, not to the level of normalization to which Adaptive Server is set.

  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.

Standards and compliance

Table 1-5 lists the ANSI SQL standards and compliance levels for Transact-SQL datatypes.

Table 1-5: ANSI SQL standards and compliance levels for Transact-SQL datatypes

Transact-SQL – ANSI SQL datatypes	Transact-SQL extensions – User-defined datatypes
• char	• binary
• varchar	• varbinary
• smallint	• bit
• int	• nchar
• bigint	• datetime
• decimal	• smalldatetime
• numeric	• bigdatetime
• float	• bigtime
• real	• tinyint
• date	• unsigned smallint
• time	• unsigned int
• double precision	• unsigned bigint
	• money
	• smallmoney
	• text
	• unitext
	• image
	• nvarchar
	• unichar
	• univarchar
	• sysname
	• longsysname
	• timestamp

Exact numeric datatypes

Use the exact numeric datatypes when you must 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 the following exact numeric datatypes to store integers: bigint, int (or integer), smallint, tinyint and each of their unsigned counterparts. 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 1-6.

Table 1-6: Integer datatypes

Datatype	Stores	Bytes of storage
bigint	Whole numbers between -2^{63} and 2^{63} - 1 (from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807, inclusive.	8
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
unsigned bigint	Whole numbers between 0 and 18,446,744,073,709,551,615	8
unsigned int	Whole numbers between 0 and 4,294,967,295	4
unsigned smallint	Whole numbers between 0 and 65,535	2

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, integer, and bigint datatypes can be preceded by an optional plus or minus sign. The tinyint datatype can be preceded by an optional plus sign.

Table 1-7 shows some valid entries for a column with a datatype of integer and indicates how isql displays these values:

Table 1-7: Valid integer values

Value entered	Value displayed
2	2
+2	2
-2	-2
2.	2
2.000	2

Table 1-8 lists some invalid entries for an integer column:

Table 1-8: 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. The numeric and decimal datatypes are identical in all respects but one: only numeric datatypes with a scale of 0 and integer datatypes 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 / log10(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 1-9 shows some valid entries for a column with a datatype of numeric(5,3) and indicates how these values are displayed by isql:

Table 1-9: 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 1-10 shows some invalid entries for a column with a datatype of numeric(5,3):

Table 1-10: 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

Transact-SQL provides the smallint, int, bigint, numeric, and decimal ANSI SQL exact numeric datatypes. The unsigned bigint, unsigned int, unsigned smallint, and tinyint type is a Transact-SQL extension.

Approximate numeric datatypes

Use the approximate numeric types, float, double precision, and real, for numeric data that can tolerate rounding. 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.

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) cannot (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 perform 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 1-11 shows the range and storage size for each approximate numeric type. isql displays only 6 significant digits after the decimal point and rounds the remainder:

Table 1-11: 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.

NaN and Inf values

“NaN” and “Inf” are special values that the IEEE754/854 floating point number standards use to represent values that are “not a number” and “infinity,” respectively. In accordance with the ANSI SQL92 standard, Adaptive Server versions 12.5 and later do not allow the insertion of these values in the database and do not allow them to be generated. In Adaptive Server versions earlier than 12.5, Open Client clients such as native-mode bcp, JDBC, and ODBC could occasionally force these values into tables.

If you encounter a NaN or an Inf value in the database, contact Sybase Customer Support with details of how to reproduce the problem.

Standards and compliance

ANSI SQL – Compliance level: The float, double precision, and real datatypes are entry-level compliant.

Money datatypes

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 1-12 summarizes the range and storage requirements for money datatypes:

Table 1-12: 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

ANSI SQL – The money and smallmoney datatypes are Transact-SQL extensions.

Timestamp datatype

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 may be confusing to other users and does 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

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

Adaptive Server has various datatypes used to store date and time values.

- date
- time
- smalldatetime
- datetime
- bigdatetime
- bigtime

The default display format for dates is “Apr 15 1987 10:23PM”. bigdatetime/bigtime types have a default display format of “Apr 15 1987 10:23:00.000000PM” You can use the convert function for other styles of date display. You can also perform some arithmetic calculations on date and time values with the built-in date functions, though Adaptive Server may round or truncate millisecond values.

- datetime columns hold dates between January 1, 1753 and December 31, 9999. datetime values are accurate to 1/300 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. Its storage size is 4 bytes: 2 bytes for the number of days after January 1, 1900, and 2 bytes for the number of minutes after midnight.
- bigdatetime columns hold dates from January 1, 0001 to December 31, 9999 and 12:00:00.000000 AM to 11:59:59.999999 PM. Its storage size is 8 bytes. The internal representation of bigdatetime is a 64 bit integer containing the number of microseconds since 01/01/0000.
- bigtime columns hold times from 12:00:00.000000 AM to 11:59:59.999999 PM. Its storage size is 8 bytes. The internal representation of bigtime is a 64 bit integer containing the number of microseconds since midnight.
- date columns hold dates from January 1, 0001 to December 31, 9999. Storage size is 4 bytes.
- time is between 00:00:00:000 and 23:59:59:999. You can use either military time or 12AM for noon and 12PM for midnight. A time value must contain either a colon or the AM or PM signifier. AM or PM may be in either uppercase or lowercase.

When entering date and time information, always enclose the time or date in single or double quotes.

Range and storage requirements

Table 1-13 summarizes the range and storage requirements for the datetime, smalldatetime, bigdatetime, bigtime, date, and time datatypes:

Table 1-13: 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
bigdatetime	January 1, 0001 to December 31, 9999	8
bigtime	12:00:00.000000AM to 11:59:59.999999PM	8
date	January 1, 0001 to December 31, 9999	4
time	12:00:00 AM to 11:59:59:999 PM	4

Entering date and time data

The datetime, smalldatetime, bigdatetime and bigtime 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.) The date datatype has only a date and the time datatype has only the time. You must enclose values in single or double quotes.

Entering the date

Dates consist of a month, day, and year and can be entered in a variety of formats for date, datetime, bigdatetime, bigtime and smalldatetime:

- 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. If you omit the date portion of a bigdatetime a default value of January 1, 0001 will be added.

Table 1-14 describes the acceptable formats for entering the date portion of a datetime or smalldatetime value:

Table 1-14: Date formats for date and time 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

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

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

The time component of a bigdatetime or bigtime value must be specified as follows:

```
hours[:minutes[:seconds[.microseconds]] [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, "15:30:20:1" means twenty seconds and one millisecond past 3:30 PM; "15:30:20.1" means twenty and one-tenth of a second past 3:30 PM. Microseconds must be expressed with a decimal point.
- If you omit the time portion of a datetime or smalldatetime value, Adaptive Server uses the default time of 12:00:00:000AM.

Displaying formats for *datetime*, *smalldatetime*, and *date* 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 1-15 lists some examples of datetime entries and their display values:

Table 1-15: Examples of datetime and date 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
Examples of date	
"1947"	Jan 1 1947
"450128"	Jan 28 2045
"520317"	Mar 17 1952

Display formats for bigdatetime and bigtime

For bigdatetime and bigtime the value displays reflects a microsecond value. bigdatetime and bigtime have default display formats that accomodate their increased precision.

- hh:mm:ss.zzzzzzAM or PM

- hh:mm:ss.zzzzzz
- mon dd yyyy
  hh:mm:ss.zzzzzzAM(PM)
- mon dd yyyy
  hh:mm:ss.zzzzzz
- yyyy-mm-dd
  hh:mm:ss.zzzzzz

The format for time must be specified as:

hours[:minutes[:seconds[.microseconds]] [*AM* | *PM*]

hours[:minutes[:seconds[number of milliseconds]] [*AM* | *PM*]

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

The seconds specification can include either a decimal portion preceded by a 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.

To store a bigdatetime or bigtime time value that includes microseconds, specify a string literal using a point. “00:00:00.1” means one tenth of a second past midnight and “00:00:00.000001” means one millionth of a second past midnight. Any value after the colon specifying fractional seconds will continue to refer to a number of milliseconds. Such as “00:00:00:5” means 5 milliseconds.

Displaying formats for *time* value

The display format for time values is “hh:mm:ss:mmmAM” (or “PM”); for example, “10:23:40:022PM.

Table 1-16: Examples of time entries

Entry	Value displayed
"12:12:00”	12:12PM
“01:23PM” or “01:23:1PM”	1:23PM
“02:24:00:001”	2:24AM

Finding 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 date or time 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 or date 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, smalldatetime, bigdatetime, bigtime, date, and time values. You cannot search for seconds or milliseconds with like and match a 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 following clause with 1 space between "May" and "2") finds all dates from May 20 through May 29, but not May 2:

```
like "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 date and time datatypes values with the built-in date functions. See "Date functions" on page 73.

Standards and compliance

ANSI SQL – Compliance level: The datetime and smalldatetime datatypes are Transact-SQL extensions. date and time datatypes are entry-level compliant.

Character datatypes

Which datatype you use for a situation depends on the type of data you are storing:

- Use the character datatypes to store strings consisting of letters, numbers, and symbols.
- Use varchar(n) and char(n) for both single-byte character sets such as us_english and for multibyte character sets such as Japanese.
- Use the unichar(n) and univarchar(n) datatypes to store Unicode characters. They are useful for single-byte or multibyte characters when you need a fixed number of bytes per character.
- Use the fixed-length datatype, nchar(n) , and the variable-length datatype, nvarchar(n), for both single-byte and multibyte character sets, such as Japanese. The difference between nchar(n) and char(n) and nvarchar(n) and varchar(n) is that both nchar(n) and nvarchar(n) allocate storage based on *n* times the number of bytes per character (based on the default character set). char(n) and varchar(n) allocate *n* bytes of storage.
- Character datatypes can store a maximum of a page size worth of data
- Use the text datatype (described in "text, image, and unitext datatypes" on page 36)—or multiple rows in a subtable—for strings longer than the char or varchar dataype allow.

unichar, *univarchar*

You can use the unichar and univarchar datatypes anywhere that you can use char and varchar character datatypes, without having to make syntax changes.

In Adaptive Server version 12.5.1 and later, queries containing character literals that cannot be represented in the server's character set are automatically promoted to the unichar datatype so you do not have to make syntax changes for data manipulation language (DML) statements. Additional syntax is available for specifying arbitrary characters in character literals, but the decision to "promote" a literal to unichar is based solely on representability.

With data definition language (DDL) statements, the syntax changes required are minimal. For example, in the create table command, the size of a Unicode column is specified in units of 16-bit Unicode values, not bytes, thereby maintaining the similarity between char(200) and unichar(200). sp_help, which reports on the lengths of columns, uses the same units. The multiplication factor (2) is stored in the new global variable *@@unicharsize*.

See Chapter 8, "Configuring Character Sets, Sort Orders, and Languages," in the *System Administration Guide* for more information about Unicode.

Length and storage size

Character variables strip the trailing spaces from strings when the variable is populated in a varchar column of a cursor.

Use *n* to specify the number of bytes of storage for char and varchar datatypes. For unichar, use *n* to specify the number of Unicode characters (the amount of storage allocated is 2 bytes per character). For nchar and nvarchar, *n* is the number of characters (the amount of storage allocated is *n* times the number of bytes per characer for the server's current default character set).

If you do not use *n* to specify the length:

- The default length is 1 byte for columns created with create table, alter table, and variables created with declare.
- The default length is 30 bytes for values created with the convert function.

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 1-17 summarizes the storage requirements of the different character datatypes:

Table 1-17: Character datatypes

Datatype	Stores	Bytes of storage
char(n)	Character	*n*
unichar(n)	Unicode character	*n***@@unicharsize* (*@@unicharsize* equals 2)
nchar(n)	National character	*n* * *@@ncharsize*
varchar(n)	Character varying	Actual number of characters entered
univarchar(n)	Unicode character varying	Actual number of characters * *@@unicharsize*
nvarchar(n)	National character varying	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 NULL values, 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.

For more information about quoted identifiers, see the section “Delimited identifiers”of the *Transact SQL User’s Guide*.

Entering Unicode characters

Optional syntax allows you to specify arbitrary Unicode characters. If a character literal is immediately preceded by U& or u& (with no intervening white space), the parser recognizes escape sequences within the literal. An escape sequence of the form \xxxx (where xxxx represents four hexadecimal digits) is replaced with the Unicode character whose scalar value is xxxx. Similarly, an escape sequence of the form \+yyyyyy is replaced with the Unicode character whose scalar value is yyyyyy. The escape sequence \\ is replaced by a single \. For example, the following is equivalent to:

```
select * from mytable where unichar_column = '五'
```

```
select * from mytable where unichar_column = U&'\4e94'
```

The U& or u& prefix simply enables the recognition of escapes. The datatype of the literal is chosen solely on the basis of representability. Thus, for example, the following two queries are equivalent:

```
select * from mytable where char_column = 'A'
```

```
select * from mytable where char_column = U&'\0041'
```

In both cases, the datatype of the character literal is char, since “A” is an ASCII character, and ASCII is a subset of all Sybase-supported server character sets.

The U& and u& prefixes also work with the double-quoted character literals and for quoted identifiers. However, quoted identifiers must be representable in the server’s character set, insofar as all database objects are identified by names in system tables, and all such names are of datatype char.

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, nchar, 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. You can use strings consisting of numbers for arithmetic after being converted to exact and approximate numeric datatypes with the convert function.

Standards and compliance

ANSI SQL – Compliance level: Transact-SQL provides the char and varchar ANSI SQL datatypes. The nchar, nvarchar, unichar, and univarchar datatypes are Transact-SQL extensions.

Binary datatypes

Use the binary datatypes, binary(n) and varbinary(n), to store raw binary data, such as pictures, in a raw binary 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.

If you do not use *n* to specify the length:

- The default length is 1 byte for columns created with create table, alter table, and variables created with declare.
- The default length is 30 bytes for values created with the convert function.

Entries of more than the maximum 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 “text, image, and unitext datatypes” on page 36.

Treatment of trailing zeros

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

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

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

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

select * from zeros


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

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

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

```
insert zeros values (0x0, 0x0,0x0, 0x0)
select * from zeros where bnot = 0x00

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

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

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

insert sample values (’002710000000ae1b’)

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

Platform dependence

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

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

For platform-independent conversions between hexadecimal strings and integers, use the inttohex and hextoint functions rather than the platform-specific convert function. For details, see “Datatype conversion functions” on page 63.

Standards and compliance

ANSI SQL – Compliance level: The binary and varbinary datatypes are Transact-SQL extensions.

bit datatype

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.

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

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.

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

Standards and compliance

ANSI SQL – Compliance level: Transact-SQL extension.

sysname and *longsysname* datatypes

sysname and longsysname are user-defined datatypes that are distributed on the Adaptive Server installation tape and used in the system tables. The definitions are:

- sysname – varchar(30) "not null"
- longsysname – varchar(255) "not null"

You can declare a column, parameter, or variable to be of types sysname and longsysname. Alternately, you can also create a user-defined datatype with a base type of sysname and longsysname, and then define columns, parameters, and variables with the user-defined datatype.

Standards and compliance

ANSI SQL – Compliance level: All user-defined datatypes, including sysname and longsysname, are Transact-SQL extensions.

text, *image*, and *unitext* datatypes

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

The variable-length unitext datatype can hold up to 1,073,741,823 Unicode characters (2,147,483,646 bytes).

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

A key distinction between text and image is that text is subject to character-set conversion if you are not using the default character set of Adaptive Server default. image is not subject to character-set conversion.

Define a text, unitext, or image column as you would any other column, with a create table or alter table statement. text, unitext, or image datatype definitions do not include lengths. text, unitext, and image columns do permit null values. Their column definition takes the form:

column_name {text | image | unitext} [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)
```

This example creates a unitext column that allows null values:

```
create table tb (ut unitext 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)
```

Adaptive Server stores text, unitext, and image data in a linked list of data pages that are separate from the rest of the table. Each text, unitext, or image page stores one logical page size worth of data (2, 4, 8, or 16K). All text, unitext, and image data for a table is stored in a single page chain, regardless of the number of text, unitext, and image columns the table contains.

You can place subsequent allocations for text, unitext, and image data pages on a different logical device with sp_placeobject.

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

You can use the partition option of the alter table command to partition a table that contains text, unitext, 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, unitext, and image columns are stored.

You can use unitext anywhere you use the text datatype, with the same semantics. unitext columns are stored in UTF-16 encoding, regardless of the Adaptive Server default character set.

Data structures used for storing *text*, *unitext*, and *image* data

When you allocate text, unitext, or image data, a 16-byte text pointer is inserted into the row you allocated. Part of this text pointer refers to a text page number at the head of the text, unitext, or image data. This text pointer is known as the first text page.

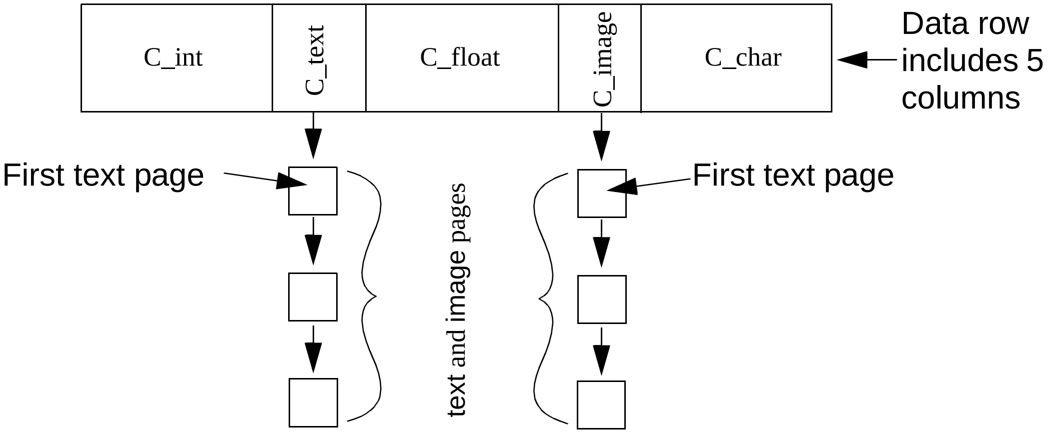
The first text page contains two parts:

- The text data page chain, which contains the text and image data and is a double-linked list of text pages
- The optional text-node structure, which is used to access the user text data

Once an first text page is allocated for text, unitext, or image data, it is never deallocated. If an update to an existing text, unitext,or image data row results in fewer text pages than are currently allocated for this text, unitext, or image data, Adaptive Server deallocates the extra text pages. If an update to text, unitext, or image data sets the value to NULL, all pages except the first text page are deallocated.

Figure 1-1 shows the relationship between the data row and the text pages.

Figure 1-1: Relationship between the text pointer and data rows



In Figure 1-1, columns c_text and c_image are text and image columns containing the pages at the bottom of the picture.

Initializing *text*, *unitext*, and *image* columns

text, unitext, 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, unitext, or image data value. It also creates a pointer in the table to the location of the text, unitext, 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 Surround text values with quotation marks and precede image values with the characters “0x”.

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

Defining unitext columns

You can define a unitext column the same way you define other datatypes, using create table or alter table statements. You do not define the length of a unitext column, and the column can be null.

This example creates a unitext column that allows null values:

```
create table tb (ut unitext null)
```

default unicode sort order defines the sort order for unitext columns for pattern matching in like clauses and in the patindex function, this is independent of the Adaptive Server default sort order.

Saving space by allowing NULL

To save storage space for empty text, unitext, 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, unitext, 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")
```

Getting information from *sysindexes*

Each table with text, unitext, 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 1-18: 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_pages(db_id(), object_id("blurbs"), indid)
    from sysindexes
    where name = "tblurbs"
```

Note The system tables poster shows a one-to-one 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 in *Reference Manual: Commands*.

Using update to replace existing text, unitext, 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.

There are restrictions for using readtext and writetext on a column defined for unitext. For more information see the “Usage” sections under readtext and writetext in the *Reference Manual: Commands*.

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*, *image*, and *unitext* columns

You cannot use text, image, or unitext columns:

- As parameters to stored procedures or as values passed to these parameters
- As local variables
- In order by clause, compute clause, 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

Selecting *text*, *unitext*, and *image* data

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

Table 1-19: text , unitext, and image global variables

Variable	Explanation
@@textptr	The text pointer of the last text, unitext, 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, unitext, or image data to be returned with a select statement. It defaults to 32K. The maximum size for *@@textsize* is 2^{31} - 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.

Converting to or from unitext

You can implicitly convert any character or binary datatype to unitext, as well as explicitly convert to and from unitext to other datatypes. The conversion result, however, is limited to the maximum length of the destination datatype. When a unitext value cannot fit the destination buffer on a Unicode character boundary, data is truncated. If you have enabled enable surrogate processing, the unitext value is never truncated in the middle of a surrogate pair of values, which means that fewer bytes may be returned after the datatype conversion. For example, if a unitext column ut in table tb stores the string "U+0041U+0042U+00c2" (U+0041 representing the Unicode character "A"), this query returns the value "AB" if the server's character set is UTF-8, because U+00C2 is converted to 2-byte UTF-8 0xc382:

```
select convert(char(3), ut) from tb
```

Table 1-20: Converting to and from unitext

These datatypes convert implicitly *to* unitext	These datatypes convert implicitly *from* unitext	These datatypes convert explicitly *from* unitext
char, varchar, unichar, univarchar, binary, varbinary, text, image	text, image	char, varchar, unichar, univarchar, binary, varbinary

The alter table modify command does not support text, image, or unitext columns to be the modified column. To migrate from a text to a unitext column:

- Use bcp out -Jutf8 out to copy text column data out
- Create a table with unitext columns
- Use bcp in -Jutf8 to insert data into the new table

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, unitext, 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, image, and unitext data uniquely identifies each row. Therefore, a table that contains text, image, and unitext data does not contain duplicate rows unless there are rows in which all text, image, and unitext data is NULL. If this is the case, the pointer has not been initialized.

Standards and compliance

ANSI SQL – Compliance level: The text, image, and unitext datatypes are Transact-SQL extensions.

Datatypes and encrypted columns

Table 1-21 lists the supported datatypes for encrypted columns, as well as the on-disk length of encrypted columns for datatypes supported for Adaptive Server version 15.0.2.

Table 1-21: Datatype length for encrypted columns

Datatype	Input data length	Encrypted column type	Max encrypted data length (no init_vector)	Actual encrypted data length (no init vector)	Max encrypted data length with init_vector	Actual encrypted data length (with init_vector)
date	4	varbinary	17	17	33	33

Datatype	Input data length	Encrypted column type	Max encrypted data length (no init_vector)	Actual encrypted data length (no init vector)	Max encrypted data length with init_vector	Actual encrypted data length (with init_vector)
time	4	varbinary	17	17	33	33
smalldatetime	4	varbinary	17	17	33	33
bigdatetime	8	varbinary	17	17	33	33
bigtime	8	varbinary	17	17	33	33
datetime	8	varbinary	17	17	33	33
smallmoney	4	varbinary	17	17	33	33
money	8	varbinary	17	17	33	33
bit	8	varbinary	17	17	33	33
bigint	8	varbinary	17	1 7	33	33
unsigned bigint	8	varbinary	17	17	33	33
unichar(10)	2 (1 unichar character)	varbinary	33	17	49	33
unichar(10)	20 (10 unichar characters)	varbinary	33	33	49	49
univarchar(20)	20 (10 unichar characters)	varbinary	49	33	65	49

text, image, and unitext datatypes are not supported for this release of Adaptive Server.

User-defined datatypes

User-defined datatypes are built from the system datatypes and from the sysname or longsysname user-defined datatypes. 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.

A user-defined datatype must be created in each database in which it will be used. 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.

Adaptive Server allows you to create user-defined datatypes, based on any system datatype, using sp_addtype. 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 and longsysname datatypes are exceptions to this rule. Though sysname and longsysname are user-defined datatypes, you can use them 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.

Use sp_rename to rename 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.

Use sp_help 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

ANSI SQL – Compliance level: 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 and time information.
Date functions	Perform computations on datetime, smalldatetime, date, and time values and their components, date parts.
Mathematical functions	Commonly needed for operations on mathematical data.

Type of function	Description
Security functions	Security-related information.
String functions	Operate on binary data, character strings, and expressions.
System functions	Retrieves special information from the database and database objects.
Text and image functions	Supply values commonly needed for operations on text, unitext, 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 on page 80	Mathematical	The absolute value of an expression.
acos on page 81	Mathematical	The angle (in radians) with a specified cosine.
ascii on page 82	String	The ASCII code for the first character in an expression.
asin on page 84	Mathematical	The angle (in radians) with a specified sine.
atan on page 85	Mathematical	The angle (in radians) with a specified tangent.
atn2 on page 86	Mathematical	The angle (in radians) with specified sine and cosine.
audit_event_name on page 89	Security	A description of an audit event
avg on page 87	Aggregate	The numeric average of all (distinct) values.
biginttohex on page 92	Datatype conversion	Returns the platform-independent hexadecimal equivalent of the specified integer.
case on page 96		Allows SQL expressions to be written for conditional values. case expressions can be used anywhere a value expression can be used.
cast on page 99	Datatype conversion	A specified value, converted to another datatype
ceiling on page 102	Mathematical	The smallest integer greater than or equal to the specified value.
char on page 104	String	The character equivalent of an integer.
charindex on page 108	String	Returns an integer representing the starting position of an expression.
char_length on page 106	String	The number of characters in an expression.
col_length on page 111	System	The defined length of a column.
col_name on page 112	System	The name of the column with specified table and column IDs.
compare on page 113	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 on page 118	Datatype conversion	The specified value, converted to another datatype or a different datetime display format.

Function	Type	Return value
cos on page 124	Mathematical	The cosine of the specified angle (in radians).
cot on page 125	Mathematical	The cotangent of the specified angle (in radians).
count on page 126	Aggregate	The number of (distinct) non-null values as an integer.
count_big on page 128	Aggregrate	The number of (distinct) non-null values as a bigint.
current_date on page 132	Date	Returns the current date.
current_time on page 133	Date	Returns the current time.
curunreservedpgs on page 134	System	The number of free pages in the specified disk piece.
data_pages on page 136	System	The number of pages used by the specified table or index.
datalength on page 140	System	The actual length, in bytes, of the specified column or string.
dateadd on page 141	Date	The date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.
datediff on page 144	Date	The difference between two date expressions.
datename on page 148	Date	The name of the specified part of a date expression.
datepart on page 150	Date	The integer value of the specified part of a date expression.
day on page 155	Date	Returns an integer that represents the day in the datepart of a specified date.
db_id on page 158	System	The ID number of the specified database.
db_name on page 160	System	The name of the database with a specified ID number.
degrees on page 162	Mathematical	The size, in degrees, of an angle with a specified number of radians.
derived_stat on page 163	System	Returns derived statistics for the specified object and index.
difference on page 168	String	The difference between two soundex values.
exp on page 169	Mathematical	The value that results from raising the constant e to the specified power.
floor on page 170	Mathematical	The largest integer that is less than or equal to the specified value.
get_appcontext on page 172	Security	Returns the value of the attribute in a specified context.
getdate on page 173	Date	The current system date and time.
hextobigint on page 181	Datatype conversion	The bigint value equivalent of a hexadecimal string
hextoint on page 182	Datatype conversion	The platform-independent integer equivalent of the specified hexadecimal string.
host_id on page 183	System	Returns the client computer’s operating system process ID for the current Adaptive Server client.

Function	Type	Return value
host_name on page 184	System	The current host computer name of the client process.
identity_burn_max on page 186		The identity_burn_max value.
index_col on page 187	System	The name of the indexed column in the specified table or view.
index_colorder on page 188	System	Returns the column order
inttohex on page 190	Datatype conversion	The platform-independent, hexadecimal equivalent of the specified integer.
isdate on page 191	Datatype conversion	Determines whether an input expression is a valid datetime value
isnumeric on page 192	Datatype conversion	Determines if an expression is a valid numeric datatype
is_quiesced on page 193	System	Indicates whether a database is in quiesce database mode. is_quiesced returns 1 if the database is quiesced and 0 if it is not.
is_sec_service_on on page 195	Security	1 if the security service is active; 0 if it is not.
isnull on page 196	System	Substitutes the value specified in *expression2* when *expression1* evaluates to NULL.
lct_admin on page 201	System	Manages the last-chance threshold.
left on page 204	String	Returns a specified number of characters on the left end of a character string.
len on page 205	String	Returns the number of characters, not the number of bytes, of a specified string expression, excluding trailing blanks.
license_enabled on page 206	System	1" if the feature's license is enabled; 0 if it is not.
list_appcontext on page 207	Security	Lists all the attributes of all the contexts in the current session.
lockscheme on page 208	Mathematical	Returns the locking scheme of the specified object as a string.
log on page 209	Mathematical	The natural logarithm of the specified number.
log10 on page 210	Mathematical	The base 10 logarithm of the specified number.
lower on page 211	String	The lowercase equivalent of the specified expression.
ltrim on page 212	String	The specified expression, trimmed of leading blanks
max on page 213	Aggregate	The highest value in a column.
min on page 215	Aggregate	The lowest value in a column.
month on page 216	Date	An integer that represents the month in the datepart of a specified date
mut_excl_roles on page 217	Security	The mutual exclusivity between two roles.

Function	Type	Return value
newid on page 218	System	Generates human-readable, globally unique IDs (GUIDs) in two different formats, based on arguments you provide.
next_identity on page 220	System	Retrieves the next identity value that is available for the next insert.
nullif on page 221		Allows SQL expressions to be written for conditional values. nullif expressions can be used anywhere a value expression can be used; alternative for a case expression.
object_id on page 227	System	The object ID of the specified object.
object_name on page 228	System	The name of the object with the specified object ID.
pagesize on page 230	Mathematical	Returns the page size, in bytes, for the specified object.
partition_id on page 232	System	Returns the partition ID of the specified data or index partition name.
partition_name on page 233	System	The explicit name of a new partition, partition_name returns the partition name of the specified data or index partition id.
partition_object_id on page 234	System	Displays the object ID for a specified partition ID and database ID.
patindex on page 235	String, Text, Unitext, and Image	The starting position of the first occurrence of a specified pattern.
pi on page 238	Mathematical	The constant value 3.1415926535897936.
power on page 239	Mathematical	The value that results from raising the specified number to a given power.
proc_role on page 240	Security	1 if the user has the correct role to execute the procedure; 0 if the user does not have this role.
pssinfo on page 242	System	Returns information from the Adaptive Server process status structure (pss)
radians on page 243	Mathematical	The size, in radians, of an angle with a specified number of degrees.
rand on page 244	Mathematical	A random value between 0 and 1, generated using the specified seed value.
replicate on page 246	String	A string consisting of the specified expression repeated a given number of times.
reserved_pages on page 250	System	The number of pages allocated to the specified table or index.
reverse on page 254	String	The specified string, with characters listed in reverse order.
right on page 255	String	The part of the character expression, starting the specified number of characters from the right.
rm_appcontext on page 257	Security	Removes a specific application context, or all application contexts.

Function	Type	Return value
role_contain on page 258	Security	1 if *role2* contains *role1*.
role_id on page 259	Security	The system role ID of the role with the name you specify.
role_name on page 260	Security	The name of a role with the system role ID you specify.
round on page 261	Mathematical	The value of the specified number, rounded to a given number of decimal places.
row_count on page 263	System	An estimate of the number of rows in the specified table.
rtrim on page 264	String	The specified expression, trimmed of trailing blanks.
set_appcontext on page 266	Security	Sets an application context name, attribute name, and attribute value for a user session, defined by the attributes of a specified application.
show_role on page 268	Security	The login's currently active roles.
show_sec_services on page 269	Security	A list of the user's currently active security services.
sign on page 270	Mathematical	The sign (+1 for positive, 0, or -1 for negative) of the specified value.
sin on page 271	Mathematical	The sine of the specified angle (in radians).
sortkey on page 272	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 on page 277	String	A 4-character code representing the way an expression sounds.
space on page 278	String	A string consisting of the specified number of single-byte spaces.
square on page 280	Mathematical	Returns the square of a specified value expressed as a float.
sqrt on page 281	Mathematical	The square root of the specified number.
str on page 289	String	The character equivalent of the specified number.
str_replace on page 291	String	Replaces any instances of the second string expression that occur within the first string expression with a third expression.
stuff on page 295	String	The string formed by deleting a specified number of characters from one string and replacing them with another string.
substring on page 297	String	The string formed by extracting a specified number of characters from another string.
sum on page 299	Aggregate	The total of the values.
suser_id	System	The server user's ID number from the syslogins system table.
suser_name on page 302	System	The name of the current server user, or the user where the server user ID is specified.
syb_quit on page 303	System	Terminates the connection.
syb_sendmsg on page 304	System	Sends a message to a User Datagram Protocol (UDP) port.
tan on page 306	Mathematical	The tangent of the specified angle (in radians).

Function	Type	Return value
tempdb_id on page 307	System	The database ID of the temporary database assigned to the specified spid
textptr on page 308	Text, Unitext, and Image	The pointer to the first page of the specified text column.
textvalid on page 309	Text and Image	1 if the pointer to the specified text column is valid; 0 if it is not.
to_unichar on page 310	String	A unichar expression having the value of the integer expression.
tran_dumpable_status on page 311	System	Returns a true/false indication of whether dump transaction is allowed.
tsequal on page 313	System	Compares timestamp values to prevent update on a row that has been modified since it was selected for browsing.
uhighsurr on page 315	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.
ulowsurr on page 316	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 on page 317	String	The uppercase equivalent of the specified string.
uscalar on page 318	String	The Unicode scalar value for the first Unicode character in an expression.
used_pages on page 319	System	The number of pages used by the specified table and its clustered index.
user on page 321	System	The name of the current server user.
user_id on page 322	System	The ID number of the specified user or the current user.
user_name on page 323	System	The name within the database of the specified user or the current user.
valid_name on page 324	System	0 if the specified string is not a valid identifier; a number other than 0 if the string is valid.
valid_user on page 325	System	1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.
year on page 350	Date	An integer that represents the year in the datepart of a specified date.

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
- count_big
- 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 (46).

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 have semantics that include the unichar datatype.

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

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 placed into a single group.

```
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
```

Example 3

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:

```
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).

You cannot use aggregate functions 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 there is no way to store the compute clause output in the resulting table.

Statistical aggregate functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the group by clause of the select statement.

Simple aggregate functions, such as sum, avg, max, min, count_big, and count are allowed only in the select list and in the having and order by clauses as well as the compute clause of a select statement. These functions summarize data over a group of rows from the database.

Adaptive Server Enterprise supports statistical aggregate functions, which permit statistical analysis of numeric data. These functions include stddev, stddev_samp, stddev_pop, variance, var_samp, and var_pop.

These functions, including stddev and variance, are true aggregate functions in that they can compute values for a group of rows as determined by the query's group by clause. As with other basic aggregate functions such as max or min, their computation ignores null values in the input. Also, regardless of the domain of the expression being analyzed, all variance and standard deviation computation uses IEEE double-precision floating-point standard.

If the input to any variance or standard deviation function is the empty set, then each function returns as its result a null value. If the input to any variance or standard deviation function is a single value, then each function returns 0 as its result.

Standard deviation and variance

The statistical aggregate functions (and their aliases) are:

- stddev_pop (also stdevp) – standard deviation of a population. Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance. See stddev_pop on page 285 for syntax and usage information.
- stddev_samp (also stdev, stddev) – standard deviation of a sample. Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance. See stddev_samp on page 287 for syntax and usage information.
- var_pop (also varp) – variance of a population. Computes the population variance of value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression from the mean of value expression, divided by the number of rows in the group. See var_pop on page 327 for syntax and usage information.
- var_samp (also var, variance) – variance of a sample. Computes the sample variance of value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference from the mean of the value expression, divided by one less than the number of rows in the group. See var_samp on page 329 for syntax and usage information.

Statistical aggregates

Statistical aggregates are similar to the avg aggregate in that:

- The syntax is:

  var_pop ([all | distinct] *expression*)

- Only expressions with numerical datatypes are valid.
- Null values do not participate in the calculation.
- The result is NULL only if no data participates in the calculation.
- The distinct or all clauses can precede the expression (the default is all).

- You can use statistical aggregates as vector aggregates (with group by), scalar aggregates (without group by), or in the compute clause.

Unlike the avg aggregate, however, the results are:

- Always of float datatype (that is, a double-precision floating-point), whereas for the avg aggregate, the datatype of the result is the same as that of the expression (with exceptions).
- 0.0 for a single data point.

Formulas

Figure 2-1: The formula for population-related statistical aggregate functions

The formula that defines the variance of the population of size *n* having mean μ (var_pop) is as follows. The population standard deviation (stddev_pop) is the positive square root of this.

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{n}$$

σ^2 = Variance

n = Population size

μ = Mean of the values x_i

Figure 2-2: The formula for sample-related statistical aggregate functions

The formula that defines an unbiased estimate of the population variance from a sample of size *n* having mean $\bar{x}$ (var_samp) is as follows. The sample standard deviation (stddev_samp) is the positive square root of this.

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

s^2 = Variance

n = Sample size

$\bar{x}$ = Mean of the values x_i

The essential difference between the two formulas is the division by *n-1* instead of *n*.

These two functions are similar, but are used for different purposes:

- var_samp – is used when you want evaluate a sample—that is, a subset—of a population as being representative of the entire population
- var_pop – is used when you have all of the data available for a population, or when *n* is so large that the difference between *n* and *n-1* is insignificant

Datatype conversion functions

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

- cast
- convert
- inttohex
- hextoint
- hextobigint
- biginttohex
- str

You can use the datatype conversion functions 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 the following:

- smallint data to datetime
- datetime data to smallint
- binary or varbinary data to smalldatetime or datetime data

Unsupported conversions result in error messages.

bigtime and bigdatetime dataype conversions

Implicit conversion are allowed between new and existing chronological datatypes.

Implicit conversion between types whose primary fields do not match may cause either data truncation, the insertion of a default value, or an error message to be raised. For example, when a bigdatetime value is converted to a date value, the time portion is truncated leaving only the date portion. If a bigtime value is converted to a bigdatetime value, a default date portion of Jan 1, 0001 is added to the new bigdatetime value. If a date value is converted to a bigdatetime value, a default time portion of 00:00:00.000000 is added to the bigdatetime value.

Implicit and explicit conversions are allowed where a decreased precision results in the loss of data.

Table 2-3 and Table 2-4 indicate whether individual datatype conversions are performed implicitly, explicitly, or are not supported.

Table 2-3: Explicit, implicit, and unsupported datatype conversions for binary – unsigned int

From	binary	varbinary	bit	[n]char	[n]varchar	datetime	smalldatetime	bigdatetime	bigtime	tinyint	smallint	unsigned smallint	int	unsigned int
binary	–	I	I	I	I	U	U	I	I	I	I	I	I	I
varbinary	I	–	I	I	I	U	U	I	I	I	I	I	I	I
bit	I	I	–	I	I	U	U	U	U	I	I	I	I	I
[n]char	I	I	E	–	I	I	I	I	I	E	E	E	E	E
[n]varchar	I	I	E	I	–	I	I	I	I	E	E	E	E	E
datetime	I	I	U	I	I	–	I	I	I	U	U	U	U	U
smalldatetime	I	I	U	I	I	I	–	I	I	U	U	U	U	U
bigdatetime	I	I	U	I	I	I	I	-	I	U	U	U	U	U
bigtime	I	I	U	I	I	I	I	I	-	U	U	U	U	U
tinyint	I	I	I	E	E	U	U	U	U	–	I	I	I	I
smallint	I	I	I	E	E	U	U	U	U	I	–	I	I	I
unsigned smallint	I	I	I	E	E	U	U	U	U	I	I	–	I	I
int	I	I	I	E	E	U	U	U	U	I	I	I	–	I
unsigned int	I	I	I	E	E	U	U	U	U	I	I	I	I	–
bigint	I	I	I	E	E	U	U	U	U	I	I	I	I	I
unsigned bigint	I	I	I	E	E	U	U		U	I	I	I	I	I
decimal	I	I	I	E	E	U	U	U	U	I	I	I	I	I
numeric	I	I	I	E	E	U	U	U	U	I	I	I	I	I
float	I	I	I	E	E	U	U	U	U	I	I	I	I	I
real	I	I	I	E	E	U	U	U	U	I	I	I	I	I
money	I	I	I	I	I	U	U	U	U	I	I	I	I	I
smallmoney	I	I	I	I	I	U	U	U	U	I	I	I	I	I
text	U	U	U	E	E	U	U	U	U	U	U	U	U	U
unitext	E	E	E	E	E	U	U	U	U	U	U	U	U	U
image	E	E	U	U	U	U	U	U	U	U	U	U	U	U
unichar	I	I	E	I	I	I	I	I	I	E	E	E	E	E
univarchar	I	I	E	I	I	I	I	I	I	E	E	E	E	E
date	I	I	U	I	I	I	U	I	U	U	U	U	U	U
time	I	I	U	I	I	I	U	I	I	U	U	U	U	U

Table 2-4: Explicit, implicit, and unsupported datatype conversions for bigint – time

From	bigint	unsigned bigint	decimal	numeric	float	real	money	smallmoney	text	unitext	image	unichar	univarchar	date	time
binary	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I
varbinary	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I
bit	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
[n]char	E	E	E	E	E	E	E	E	I	I	I	I	I	I	I
[n]varchar	E	E	E	E	E	E	E	E	I	I	I	I	I	I	I
datetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
smalldatetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
bigdatetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
bigtime	U	U	U	U	U	U	U	U	U	U	U	I	I	U	I
tinyint	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
smallint	I	I	I	I	I	I	I	I	U	U	U	U	E	U	U
unsigned smallint	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
int	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
unsigned int	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
bigint	–	I	I	I	I	I	I	I	U	U	U	E	E	U	U
unsigned bigint	I	–	I	I	I	I	I	I	U	U	U	E	E	U	U
decimal	I	I	–	I	I	I	I	I	U	U	U	E	E	U	U
numeric	I	I	I	–	I	I	I	I	U	U	U	E	E	U	U
float	I	I	I	I	–	I	I	I	U	U	U	E	E	U	U
real	I	I	I	I	I	–	I	I	U	U	U	E	E	U	U
money	I	I	I	I	I	I	–	I	U	U	U	E	E	U	U
smallmoney	I	I	I	I	I	I	I	–	U	U	U	E	E	U	U
text	U	U	U	U	U	U	U	U	–	I	U	E	E	U	U
unitext	U	U	U	U	U	U	U	U	I	–	I	U	U	U	U
image	U	U	U	U	U	U	U	U	U	I	–	E	E	U	U
unichar	E	E	E	E	E	E	E	E	I	I	I	–	I	I	I
univarchar	E	E	E	E	E	E	E	E	I	I	I	I	–	I	I
date	U	U	U	U	U	U	U	U	U	U	U	I	I	–	I
time	U	U	U	U	U	U	U	U	U	U	U	I	I	I	–

Datatype conversion

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 arithabortnumeric_truncation is on; implicit conversion allowed otherwise.
- U – unsupported conversion.
- – conversion of a datatype to itself. These conversions are allowed, but are meaningless.

Converting character data to a noncharacter type

You can convert character data to a noncharacter 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, Adaptive 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 and time data

Implicit conversions between unichar/univarchar and datetime/smalldatetime are supported.

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 and unitext 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.
```

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

Note The str function may be preferable to convert or cast when making conversions, because it provides more control over conversions and avoids errors.

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* and *time* information

Data that is recognizable as a date can be converted to datetime, smalldatetime, date, or time. 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.

When converting date data to a character type, use style numbers 1 through 7 (101 through 107) or 10 through 12 (110 through 112) in Table 2-7 on page 119 to specify the display format. The default value is 100 (mon dd yyyy hh:miAM (or PM)). If date data is converted to a style that contains a time portion, that time portion reflects the default value of zero.

When converting time data to a character type, use style number 8 or 9 (108 or 109) to specify the display format. The default is 100 (mon dd yyyy hh:miAM (or PM)). If time data is converted to a style that contains a date portion, the default date of Jan 1, 1900 is displayed.

Converting between numeric types

You can convert data from one numeric type to another. Errors can occur if the new type is an exact numeric with precision or scale that is not sufficient to hold the data.

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.

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. You can omit optional arith_overflow keyword 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 ANSI SQL 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 (little-endian) and 256 on machines that consider byte 0 least significant (big-endian).

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. Use the hextobigint function for platform-independent conversion of hexadecimal strings to 64-bit integers, and the biginttohex function for platform-independent conversion of 64-bit 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),0x0000000000000000006b14bd1e6eea000000000000000000000000000000)

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

This example converts the same numeric data back to binary:

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

--------------------------------------------------------------
0x0000000000000000006b14bd1e6eea000000000000000000000000000000
```

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 NULL to NOT NULL and NOT NULL to NULL.

Date functions

The date functions manipulate values of the datatypes datetime, bigtime, bigdatetime, smalldatetime, date or time.

You can use date functions 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 date for dates from January, 1, 0001 to January 1, 9999. date 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” on page 63 and “Date and time datatypes” on page 20 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).

The date datatype can cover dates from January 1, 0001 to January 1, 9999.

Date parts

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

Date part	Abbreviation	Values
year	yy	1753 – 9999 (datetime) 1753 – 2079 (smalldatetime) 0001 – 9999 (bigdatetime)
quarter	qq	1 – 4
month	mm	1 – 12
week	wk	1 – 54
day	dd	1 – 31
dayofyear	dy	1 – 366

Date part	Abbreviation	Values
weekday	dw	1 – 7 (Sun. – Sat.)
hour	hh	0 – 23
minute	mi	0 – 59
second	ss	0 – 59
millisecond	ms	0 – 999
microseconds	us	0-999999

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.

For datetime, smalldatetime and time values 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 inserting datetime or time data, as these datatypes have a granularity of 1/300th of a second rather than 1/1000th of a second. You can use the time datatype for time information.

For bigdatetime and bigtime time values, microseconds must be preceded by a decimal point and represent fractional seconds.

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	• cos	• log	• rand
• acos	• cot	• log10	• round
• asin	• degrees	• pagesize	• sign
• atan	• exp	• pi	• sin
• atn2	• floor	• power	• sqrt
• ceiling	• lockscheme	• radians	• 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.

Security functions

Security functions return security-related information.

The security functions are:

- is_sec_service_on
- show_sec_services
- get_appcontext
- list_appcontext
- set_appcontext
- rm_appcontext
- show_role
- proc_role
- role_contain
- role_id
- role_name

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
- str_replace
- stuff
- substring
- to_unichar
- uhighsurr
- ulowsurr
- upper
- uscalar

You can nest string functions and use them 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. This limit is independent of the server's page size. In Transact-SQL string functions and string variables, literals can be as large as 16K even on a 2K page size.

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

```
select replicate("a", 16383) + replicate("B", 4000)
```

This shows that the total length would be 20383, but the result string is restricted to 16K.

System functions

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

- col_length
- col_name
- curunreservedpgs
- data_pages
- datalength
- db_id
- db_name
- host_id
- host_name
- index_col
- is_quiesced
- isnull
- object_id
- object_name
- reserved_pages
- row_count
- show_role
- suser_id
- suser_name
- tempdb_id
- tran_dumpable_status
- tsequal
- used_pages
- 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, unitext, and image columns

text, unitext, 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 clause, except with the keyword like
- In joins

In triggers, both the inserted and deleted text values reference the new value; you cannot reference the old value.

Text and image functions

Text and image functions operate on text, image, and unitext 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, image, and unitext data that is retrieved by a select statement.

You can use the patindex text function on text, image, and unitext columns and can consider it on a text and image function.

You can use the datalength function to display the length of data in text, image, and unitext columns.

User-defined SQL functions

You can include these in a scalar function:

- declare statements to define data variables and cursors that are local to the function.
- Assigned values to objects local to the function (for example, assigning values to scalar and variables local to a table with select or set commands).

- Cursor operations that reference local cursors that are declared, opened, closed, and deallocated in the function.
- Control-of-flow statements.
- set options (only valid in the scope of the function).

Adaptive Server does not allow fetch statements in a scalar function that return data to the client. You cannot include :

- select or fetch statements that returns data to the client.
- insert, update, or delete statements.
- Utility commands, such as dbcc, dump and load commands.
- print statements
- Statement that references rand, rand2, getdate, or newid.

You can include select or fetch statements that assign values only to local variable.

abs

Description
Returns the absolute value of an expression.

Syntax
abs(*numeric_expression*)

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

Examples
Returns the absolute value of -1:

```
select abs(-1)

-----------
          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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute abs.

See also
“Mathematical functions” on page 74 for general information about mathematical functions.

Functions ceiling, floor, round, sign

acos

Description: Returns the angle (in radians) with a specified cosine.

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: Returns the angle where the cosine is 0.52:

```
select acos(0.52)

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

Usage: acos, a mathematical function, returns the angle (in radians) where the cosine is the specified value.

Standards: ANSI SQL – Compliance level: Transact-SQL extension.

Permissions: Any user can execute acos.

See also: "Mathematical functions" on page 74 for general information about mathematical functions.

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 — Returns the author's last names and the ACSII codes for the first letters in their last names, if the ASCII code is less than 70:

```
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
```

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.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute ascii.

See also — For general information about string functions, see "String functions" on page 76.

Functions char, to_unichar

asehostname

Description Returns the physical or virtual host on which Adaptive Server is running.

Syntax asehostname

Parameters

None.

Examples Returns the Adaptive Server host name:

```
select asehostname()

-----------------------------------
linuxkernel.sybase.com
```

Standards SQL/92 and SQL/99 compliant

Permissions Only users with the sa_role can execute asehostname.

asin

Description — Returns the angle (in radians) with a specified sine.

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) with a sine of the specified value.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute asin.

See also — "Mathematical functions" on page 74 for general information about mathematical functions.

Functions degrees, radians, sin

atan

Description — Returns the angle (in radians) with a specified tangent.

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) of a tangent with the specified value.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute atan.

See also — “Mathematical functions” on page 74 for general information about mathematical functions.

Functions atn2, degrees, radians, tan

atn2

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

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.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute atn2.

See also — "Mathematical functions" on page 74 for general information about mathematical functions.

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 359.

Examples

Example 1 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:

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

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

Example 2 Used with a group by clause, the aggregate functions produce single values for each group, rather than for the entire table. This statement produces summary values for each type of book:

```
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
```

Example 3 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:

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

 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.
- When you average (signed or unsigned) int, smallint, tinyint data , Adaptive Server returns the result as an int value. When you average (signed or unsigned) bigint data, Adaptive Server returns the result as a bigint value. To avoid overflow errors in DB-Library programs, declare variables used for resultrs appropriately.
- You cannot use avg with the binary datatypes.
- Since the average value is only defined on numeric datatypes, using avg Unicode expressions generates an error.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute avg.

See also

For general information about aggregate functions, see “Aggregate functions” on page 54.

Functions max, min

audit_event_name

Description

Returns a description of an audit event.

Syntax

audit_event_name(*event_id*)

Parameters

event_id
is the number of an audit event.

Examples

Example 1 Queries the audit trail for table creation events:

```
select * from audit_data where audit_event_name(event) = "Create Table"
```

Example 2 Obtains current audit event values. See the Usage section below for a complete list of audit values and their descriptions.

```
create table #tmp(event_id int, description varchar(255))
go
declare @a int
select @a=1
while (@a<120)
begin
      insert #tmp values (@a, audit_event_name(@a))
      select @a=@a + 1
end
select * from #tmp
go
```

```
-----------------------------------------
event_id    description
---------   -------------------
      1     Ad hoc Audit Record
      2     Alter Database
      ...
      104   Create Index
      105   Drop Index
```

Usage

The following lists the ID and name of each of the audit events:

1 Ad Hoc Audit record
2 Alter Database
3 Alter table
4 BCP In
5 NULL
6 Bind Default
7 Bind Message
8 Bind Rule
9 Create Database
10 Create Table
11 Create Procedure
12 Create Trigger
13 Create Rule
14 Create Default
15 Create Message
16 Create View
17 Access To Database
18 Delete Table
19 Delete View
20 Disk Init
21 Disk Refit
22 Disk Reinit
23 Disk Mirror
24 Disk Unmirror
25 Disk Remirror
26 Drop Database
27 Drop Table
28 Drop Procedure
29 Drop Trigger
30 Drop Rule
31 Drop Default
32 Drop Message
33 Drop View
34 Dump Database
35 Dump Transaction
36 Fatal Error
37 Nonfatal Error
38 Execution Of Stored Procedure
39 Execution Of Trigger
40 Grant Command
41 Insert Table
42 Insert View
43 Load Database
44 Load Transaction
45 Log In
46 Log Out
47 Revoke Command
48 RPC In
49 RPC Out
50 Server Boot
51 Server Shutdown
52 NULL
53 NULL
54 NULL
55 Role Toggling
56 NULL
57 NULL
58 NULL
59 NULL
60 NULL
61 Access To Audit Table
62 Select Table
63 Select View
64 Truncate Table
65 NULL
66 NULL
67 Unbind Default
68 Unbind Rule
69 Unbind Message
70 Update Table
71 Update View
72 NULL
73 Auditing Enabled
74 Auditing Disabled
75 NULL
76 SSO Changed Password
79 NULL
80 Role Check Performed
81 DBCC Command
82 Config
83 Online Database
84 Setuser Command
85 User-defined Function Command
86 Built-in Function
87 Disk Release
88 Set SSA Command
90 Connect Command
91 Reference
92 Command Text
93 JCS Install Command
94 JCS Remove Command
95 Unlock Admin Account
96 Quiesce Database Command
97 Create SQLJ Function
98 Drop SQLJ Function
99 SSL Administration
100 Disk Resize
101 Mount Database
102 Unmount Database
103 Login Command
104 Create Index
105 Drop Index
106 NULL
107 NULL
108 NULL
109 NULL
110 Deploy UDWS
111 Undeploy UDWS
115 Password Administration

Note Adaptive Server does not log events if audit_event_name returns NULL.

Standards

ANSI SQL – compliance level: Transact-SQL extension.

Permissions

Any user can execute audit_event_name.

See also

Commands select, sp_audit

authmech

Description

Determines what authentication mechanism is used by a specified logged in server process ID.

Syntax

authmech ([*spid*])

Examples

Example 1 Returns the authentication mechanism for server process ID 42, whether KERBEROS, LDAP, or any other mechanism:

```
select authmech(42)
```

Example 2 Returns the authentication mechanism for the current login's server process ID:

```
select authmech()
```

or

```
select authmech(0)
```

Example 3 Prints the authentication mechanism used for each login session:

```
select suid, authmech(spid)
    from sysprocesses where suid!=0
```

Usage

- This function returns output of type varchar from one optional argument.
- If the value of the server process ID is 0, the function returns the authentication method used by the server process ID of the current client session.
- If no argument is specified, the output is the same as if the value of the server process ID is 0.
- Possible return values include `ldap`, `ase`, `pam`, and NULL.

Permissions

Any user can execute authmech to query a current personal session. You must have sso_role privileges to query the details of another user's session.

biginttohex

Description

Returns the platform-independent 8 byte hexadecimal equivalent of the specified integer expression.

Syntax

biginttohex (*integer_expression*)

Parameters

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

Examples

Converts the big integer -9223372036854775808 to a hexadecimal string:

```
1> select biginttohex(-9223372036854775808)
2> go
----------------
8000000000000000
```

Usage

- biginttohex, a datatype conversion function, returns the platform-independent hexadecimal equivalent of an integer, without a "0x" prefix.
- Use the biginttohex function for platform-independent conversions of integers to hexadecimal strings. biginttohex accepts any expression that evaluates to a bigint. It always returns the same hexadecimal equivalent for a given expression, regardless of the platform on which it is executed.

See also

Functions convert, hextobigint, hextoint, inttohex

bintostr

Description

Converts a sequence of hexadecimal digits to a string of its equivalent alphanumeric characters or varbinary data.

Syntax

select bintostr(*sequence of hexadecimal digits*)

Parameters

sequence of hexadecimal digits
is the sequence of valid hexadecimal digits, consisting of [0 – 9], [a – f] and [A – F], and which is prefixed with "0x".

Examples

Example 1 Converts the hexadecimal sequence of "0x723ad82fe" to an alphanumeric string of the same value:

```
1> select bintostr(0x723ad82fe)
2> go

-----------------------------------
0723ad82fe
```

In this example, the in-memory representation of the sequence of hexadecimal digits and its equivalent alphanumeric character string are:

Hexadecimal digits (5 bytes)

0	7	2	3	a	d	8	2	f	e

Alphanumeric character string (9 bytes)

0	7	2	3	a	d	8	2	f	e

The function processes hexadecimal digits from right to left. In this example, the number of digits in the input is odd. For this reason, the alphanumeric character sequence has a prefix of "0" and is reflected in the output.

Example 2 Converts the hexadecimal digits of a local variable called *@bin_data* to an alphanumeric string equivalent to the value of "723ad82fe":

```
declare @bin_data varchar(30)
select @bin_data = 0x723ad82fe
select bintostr(@bin_data)
go

----------
0723ad82fe
```

Usage

- Any invalid characters in the input results in null as the output.
- The input must be valid varbinary data.
- A NULL input results in NULL output.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute bintostr.

See also

Functions strtobin

cache_usage

Description

Returns cache usage as a percentage of all objects in the cache to which the table belongs.

Syntax

cache_usage(*table_name*)

Parameters

table_name
is the name of a table. The name can be fully qualified (that is, it can include the database and owner name).

Examples

Example 1 Returns percentage of the cache used by the titles tables:

```
select cache_usage("titles")
----------------------------
                     98.876953
```

Example 2 Retuns, from the master database, the percentage of the cache used by the authors tables

```
select cache_usage ("pubs2..authors")
------------------------------------
                             98.876953
```

Usage

- cache_usage provides cache usage as percentage across all the pools of the cache.
- cache_usage does not provide any information on how much cache the current object is using, and does not provide information for cache usages of indexes if they are bound to different cache.
- In cluster environments – cache_usage provides cache usage of the cache the object is bound to in current node.

Permissions

Any user can execute cache_usage.

case

Description

Supports conditional SQL expressions; can be used anywhere a value expression can be used.

Syntax

case
    when *search_condition* then *expression*
    [when *search_condition* then *expression*]...
    [else *expression*]
end

case and values syntax:

case *expression*
    when *expression* then *expression*
    [when *expression* then *expression*]...
    [else *expression*]
end

Parameters

case
begins the case expression.

when
precedes the search condition or the expression to be compared.

search_condition
is used to set conditions for the results that are selected. Search conditions for case expressions are similar to the search conditions in a where clause. Search conditions are detailed in the *Transact-SQL User's Guide*.

then
precedes the expression that specifies a result value of case.

expression
is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" on page 359.

Examples

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

```
select au_lname, postalcode,
    case
            when postalcode = "94705"
                then "Berkeley Author"
            when postalcode = "94609"
                then "Oakland Author"
            when postalcode = "94612"
                then "Oakland Author"
```

```
                when postalcode = "97330"
                    then "Corvallis Author"
            end
    from authors
```

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

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

Yuo can also use the following format to produce the same result, since coalesce is an abbreviated form of a case expression:

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

Example 3 Selects the *titles* and *type* from the titles table. If the book type is UNDECIDED, nullif returns a NULL value:

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

You can also use the following format to produce the same result, since nullif is an abbreviated form of a case expression:

```
select title,
        case
            when type = "UNDECIDED" then NULL
            else type
        end
from titles
```

Example 4 Produces an error message, because at least one expression must be something other than the null keyword:

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

```
All result expressions in a CASE expression must not be NULL.
```

Example 5 Produces an error message, because at least two expressions must follow coalesce:

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

```
A single coalesce element is illegal in a COALESCE expression.
```

Usage

- case expression simplifies standard SQL expressions by allowing you to express a search condition using a when...then construct instead of an if statement.
- case expressions can be used anywhere an expression can be used in SQL.
- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in "Datatypes of mixed-mode expressions" on page 7 in. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

case permission defaults to all users. No permission is required to use it.

See also

Commands coalesce, nullif, if...else, select, where clause

cast

Description

Returns the specified value, converted to another datatype. cast can change the nullability of the source expression, and uses the default format for date and time datatypes.

Syntax

cast (*expression* as *datatype* [(*length* | *precision*[, *scale*])])

Parameters

expression
is the value to be converted from one datatype or date format to another. It includes columns, constants, functions, any combination of constants, and functions that are connected by arithmetic or bitwise operators orsubqueries.

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 datatype, the default length of 30 Unicode values is used if no length is specified.

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 character types and 30 bytes for 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.

Examples

Example 1 Converts the date into a more readable datetime format:

```
select cast("01/03/63" as datetime)
go

 --------------------------
        Jan  3 1963 12:00AM

(1 row affected)
```

Example 2 Converts the total_sales column in the title database to a 12-character column:

```
select title, cast(total_sales as char(12))
```

Usage

- For more information about datatype conversion, see "Datatype conversion functions" on page 63.
- cast 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. You can use null or not null with select into to create a new table and change the datatype and nullability of existing columns in the source table.
- You can use cast to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes that 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.
- You can use unichar expressions as a destination datatype, or they can be converted to another datatype. unichar expressions can be converted either explicitly between any other datatype supported by the server, or implicitly.
- If you do not specify length 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, an error message appears.

Implicit conversion

Implicit conversion between types when the primary fields do not match may cause data truncation, the insertion of a default value, or an error message to be raised. For example, when a datetime value is converted to a date value, the time portion is truncated, leaving only the date portion. If a time value is converted to a datetime value, a default date portion of Jan 1, 1900 is added to the new datetime value. If a date value is converted to a datetime value, a default time portion of 00:00:00:000 is added to the datetime value.

```
DATE -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
TIME -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME -> DATE
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME  -> TIME
```

Explicit conversion

If you attempt to explicitly convert a date to a datetime, and the value is outside the datetime range such as "Jan 1, 1000" the conversion is not allowed and an informative error message is raised.

```
DATE -> UNICHAR, UNIVARCHAR
```

```
TIME -> UNICHAR, UNIVARCHAR
UNICHAR, UNIVARCHAR -> DATE
UNICHAR, UNIVARCHAR -> TIME
```

Conversions involving Java classes

- When Java is enabled in the database, you can use cast 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.

Standards

ANSI SQL – Compliance level: ANSI compliant.

Permissions

Any user can execute cast.

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 with a datatype is exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types.

Examples

Example 1 Returns a value of 124:

```
select ceiling(123.45)
```

```
124
```

Example 2 Returns a value of -123:

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

```
-123
```

Example 3 Returns a value of 24.000000:

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

```
24.000000
```

Example 4 Returns a value of -123.000000:

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

```
-123.000000
```

Example 5 Returns a value of 124.00

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

```
124.00
```

Example 6 Returns values of “discount” from the salesdetail table where title_id is the value “PS3333”:

```
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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute ceiling.

See also

For general information about mathematical functions, see "Mathematical functions" on page 74.

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.

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 — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute char.

See also — For general information about string functions, see "String functions" on page 76.

Functions ascii, str

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 unitext, unichar, and univarchar columns, char_length returns the number of Unicode values (16-bit), with one surrogate pair counted as two Unicode values. For example, this is what is returned if a unitext column ut contains row value U+0041U+0042U+d800dc00:

```
select char_length(ut) from unitable
------------
4
```

- For multibyte character sets, the number of characters in the expression is usually fewer 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 76.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute char_length.

See also

Function datalength

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, binary, or varbinary.

Examples

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

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

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

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 *expression2* is NULL, returns 0.
- 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

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute charindex.

See also

For general information about string functions, see "String functions" on page 76.

Function patindex

coalesce

Description

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

Syntax

coalesce(*expression*, *expression* [, *expression*]...)

Parameters

coalesce
: evaluates the listed expressions and returns the first non-null value. If all expressions are null, coalesce returns NULL.

expression
: is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see “Expressions” on page 359.

Examples

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

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

Example 2 An alternative way of writing the previous example:

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

Usage

- coalesce expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a when...then construct.
- You can use coalesce expressions anywhere an expression in SQL.
- At least one result of the coalesce expression must return a non-null value. This example produces the following error message:

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

```
All result expressions in a CASE expression must not be NULL.
```

- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in "Datatypes of mixed-mode expressions" on page 7. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.
- coalesce is an abbreviated form of a case expression. Example 2 describes an alternative way of writing the coalesce statement.
- coalesce must be followed by at least two expressions. This example produces the following error message:

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

```
A single coalesce element is illegal in a COALESCE expression.
```

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute coalesce.

See also — **Commands** case, nullif, select, if...else, where clause

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

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

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

 x
----
   80
```

Usage

- col_length, a system function, returns the defined length of column.
- For general information about system functions, see “System functions” on page 77.
- To find the actual length of the data stored in each row, use datalength.
- For text, unitext, 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

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute col_length.

See also

Function datalength

col_name

Description

Returns the name of the column where the table and column IDs are specified, and can be up to 255 bytes in length.

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 77.

Standards

ANSI SQL – Compliance 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_expression1*
are the character expressions to compare to *char_expression2* or *uchar_expression 2*.

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

char_expression1 and *char_expression2* can be:

- 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:

- 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 2-6 on page 116 shows the valid values.

collation_ID
is an integer constant or a variable that specifies the collation to use. Table 2-6 on page 116 shows the valid values.

Examples

Example 1 Compares aaa and bbb:

```
1> select compare ("aaa","bbb")
2> go

 -----------
          -1
(1 row affected)
```

Alternatively, you can also compare aaa and bbb using this format:

```
1> select compare (("aaa"),("bbb"))
2> go

 -----------
          -1
(1 row affected)
```

Example 2 Compares aaa and bbb and specifies binary sort order:

```
1> select compare ("aaa","bbb","binary")
2> go

 -----------
          -1
(1 row affected)
```

Alternatively, you can compare aaa and bbb using this format, and the collation ID instead of the collation name:

```
1> select compare (("aaa"),("bbb"),(50))
2> go

 -----------
          -1
(1 row affected)
```

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*.
- compare can generate up to six bytes of collation information for each input character. Therefore, the result from using compare may exceed the length limit of the varbinary datatype. If this happens, the result is truncated to fit. Adaptive Server issues a warning message, but the query or transaction that contained the compare function continues to run. Since this limit is dependent on the logical page size of your server, truncation removes result bytes for each input character until the result string is less than the following for DOL and APL tables:

Table 2-5: Maximum row and column length—APL and DOL

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

- Both *char_expression1*, *uchar_expression1*, and *char_expression2*, *uchar_expression2* must be characters that are encoded in the server's default character set.
- *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 0.
  - If *char_expression1* or *uchar_expression 1* is empty, the function returns -1.

  The compare function does not equate empty strings and strings containing only spaces. 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 do not compare equally.
- If either *char_expression1*, *uchar_expression1*; or *char_expression2*, *uchar_expression2* is NULL, then the result is 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 2-6 lists the valid values for *collation_name* and *collation_ID*.

Table 2-6: Collation names and IDs

Description	Collation name	Collation ID
Deafult Unicode multilingual	default	20
Thai dictionary order	thaidict	21
ISO14651 standard	iso14651	22
UTF-16 ordering – matches UTF-8 binary ordering	utf8bin	24
CP 850 Alternative – no accent	altnoacc	39
CP 850 Alternative – lowercase first	altdict	45
CP 850 Western European – no case preference	altnocsp	46
CP 850 Scandinavian – dictionary ordering	scandict	47
CP 850 Scandinavian – case-insensitive with preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
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 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-9 Turkish dictionary	turdict	72
ISO 8859-9 Turkish no accents	turknoac	73
ISO 8859-9 Turkish no case	turknocs	74
CP932 binary ordering	cp932bin	129
Chinese phonetic ordering	dynix	130
GB2312 binary ordering	gb2312bn	137
Common Cyrillic dictionary	cyrdict	140
Turkish dictionary	turdict	155

Description	Collation name	Collation ID
EUCKSC binary ordering	euckscbn	161
Chinese phonetic ordering	gbpinyin	163
Russian dictionary ordering	rusdict	165
SJIS binary ordering	sjisbin	179
EUCJIS binary ordering	eucjisbn	192
BIG5 binary ordering	big5bin	194
Shift-JIS binary order	sjisbin	259

Standards

ANSI SQL – Compliance 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 datatype, 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 2-7 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.

When converting date data to a character type, use style numbers 1 through 7 (101 through 107) or 10 through 12 (110 through 112) in Table 2-7 to specify the display format. The default value is 100 (mon dd yyyy hh:miAM (or PM)). If date data is converted to a style that contains a time portion, that time portion reflects the default value of zero.

When converting time data to a character type, use style number 8 or 9 (108 or 109) to specify the display format. The default is 100 (mon dd yyyy hh:miAM (or PM)). If time data is converted to a style that contains a date portion, the default date of Jan 1, 1900 is displayed.

Table 2-7: Date format conversions using the style parameter

Without century (yy)	**With century (yyyy)**	**Standard**	**Output**
-	0 or 100	Default	*mon dd yyyy hh:mm* AM (or PM)
1	101	USA	*mm/dd/yy*
2	2	SQL standard	*yy.mm.dd*
3	103	English/French	*dd/mm/yy*
4	104	German	*dd.mm.yy*
5	105		*dd-mm-yy*
6	106		*dd mon yy*
7	107		*mon dd, yy*
8	108		*HH:mm:ss*
-	9 or 109	Default + milliseconds	*mon dd yyyy hh:mm:ss* AM (or PM)
10	110	USA	*mm-dd-yy*
11	111	Japan	*yy/mm/dd*
12	112	ISO	*yymmdd*
13	113		*yy/dd/mm*
14	114		*mm/yy/dd*

Key “mon” indicates a month spelled out, “mm” the month number or minutes. “HH ”indicates a 24-hour clock value, “hh” a 12-hour clock value. The last row, 23, includes a literal “T” to separate the date and time portions of the format.

Without century (yy)	**With century (yyyy)**	**Standard**	**Output**
14	114		*hh:mi:ss:mmm*AM(or PM)
15	115		*dd/yy/mm*
-	16 or 116		*mon dd yyyy HH:mm:ss*
17	117		*hh:mmAM*
18	118		*HH:mm*
19			*hh:mm:ss:zzzAM*
20			*hh:mm:ss:zzz*
21			*yy/mm/dd HH:mm:ss*
22			*yy/mm/dd HH:mm* AM (or PM)
23			*yyyy-mm-ddTHH:mm:ss*

Key "mon" indicates a month spelled out, "mm" the month number or minutes. "HH "indicates a 24-hour clock value, "hh" a 12-hour clock value. The last row, 23, includes a literal "T" to separate the date and time portions of the format.

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 Converts the current date to style 3, dd/mm/yy:

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

Example 4 If the value pubdate can be null, you must use varchar rather than char, or errors may result:

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

Example 5 Returns the integer equivalent of the string "0x00000100". Results can vary from one platform to another:

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

Example 6 Returns the platform-specific bit pattern as a Sybase binary type:

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

Example 7 Returns 1, the bit string equivalent of $1.11:

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

Example 8 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:

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

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 63.
- convert – returns the specified value, converted to another datatype or a different datetime display format. When converting from unitext to other character and binary datatypes, the result is limited to the maximum length of the destination datatype. If the length is not specified, the converted value has a default size of 30 bytes. If you are using enabled enable surrogate processing, a surrogate pair is returned as a whole. For example, this is what is returned if you convert a unitext column that contains data U+0041U+0042U+20acU+0043 (stands for "AB €") to a UTF-8 varchar(3) column:

  ```
  select convert(varchar(3), ut) from untable
  ---
  AB
  ```
- 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).

  The result is an undefined value if:

  - The expression being converted is to a not null result.
  - The expression's value is null.

  Use the following select statement to generate a known non-NULL value for predictable results:

```
select convert(int not null isnull(col2, 5)) from table1
```

- 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.
- You can use unichar expressions as a destination datatype or you can convert them to another datatype. unichar expressions can be converted either explicitly between any other datatype supported by the server, or implicitly.
- If you do not specify the length 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, an error message appears.

Implicit conversion

Implicit conversion between types when the primary fields do not match may cause data truncation, the insertion of a default value, or an error message to be raised. For example, when a datetime value is converted to a date value, the time portion is truncated, leaving only the date portion. If a time value is converted to a datetime value, a default date portion of Jan 1, 1900 is added to the new datetime value. If a date value is converted to a datetime value, a default time portion of 00:00:00:000 is added to the datetime value.

```
DATE -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
TIME -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME -> DATE
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME  -> TIME
```

Explicit conversion

If you attempt to explicitly convert a date to a datetime and the value is outside the datetime range, such as "Jan 1, 1000" the conversion is not allowed and an informative error message is raised.

```
DATE -> UNICHAR, UNIVARCHAR
TIME -> UNICHAR, UNIVARCHAR
UNICHAR, UNIVARCHAR -> DATE
UNICHAR, UNIVARCHAR -> TIME
```

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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute convert.

See also

Documents *Java in Adaptive Server Enterprise* for a list of allowed datatype mappings and more information about datatype conversions involving Java classes.

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 74.

Standards — ANSI SQL – Compliance 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 74.

Standards
ANSI SQL – Compliance 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 as an integer.

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 359.

Examples

Example 1 Finds the number of different cities in which authors live:

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

Example 2 Lists the types in the titles table, but eliminates the types that include only one book or none:

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

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 54.
- 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*).
- You can use count 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 ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute count.

See also **Commands** compute clause, group by and having clauses, select, where clause

count_big

Description

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

Syntax

count_big([all | distinct] *expression*)

Parameters

all

applies count_big to all values. all is the default.

distinct

eliminates duplicate values before count_big 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.

Examples

Finds the number of occurances of *name* in systypes:

```
1> select count_big(name) from systypes
2> go
--------------------
42
```

Usage

- count_big, an aggregate function, finds the number of non-null values in a column.
- When distinct is specified, count_big finds the number of unique non-null values. Null values are ignored when counting.
- count_big(*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_big(*) finds the number of rows. count_big(*) 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_big(*) 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_big(*column_name*).
- You can use count_big as an existence check in a subquery. For example:

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

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

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

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute count_big.

See also

Commands compute clause, group by and having clauses, select, where clause

current_bigdatetime

Description

Returns a bigtime value representing the current time with microcecond precision. The accuracy of the current time portion is limited by the accuracy of the system clock.

Syntax

current_bigdatetime()

Parameters

None.

Examples

Example 1 Find the current bigdatetime:

```
select current_bigdatetime())
------------------------------
Nov 25 1995 10:32:00.010101AM
```

Example 2 Find the current bigdatetime:

```
select datepart(us, current_bigdatetime())
------------------------------
010101
```

Usage

Finds the current date as it exists on the server.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

Permissions

Any user can execute current_date.

See also

Datatypes Date and time datatypes

Commands select, where clause

Functions dateadd, datediff, datepart, datename, current_bigtime

current_bigtime

Description

Returns a bigtime value representing the current time with microcecond precision. The accuracy of the current time portion is limited by the accuracy of the system clock.

Syntax

current_bigtime()

Parameters

None.

Examples

Example 1 Finds the current bigtime:

```
select current_bigtime())
------------------------------
10:32:00.010101AM
```

Example 2 Finds the current bigtime:

```
select datepart(us, current_bigtime())
------------------------------
01010
```

Usage

Finds the current date as it exists on the server.

Standards

ANSI SQL – Compliance level: Entry-level compliant.

Permissions

Any user can execute current_date.

See also

Datatypes Date and time datatypes

Commands select, where clause

Functions dateadd, datediff, datepart, datename, current_bigdatetime

current_date

Description Returns the current date.

Syntax current_date()

Parameters None.

Examples **Example 1** Identifies the current date with datename:

```
1> select datename(month, current_date())
2> go

------------------------------
August
```

Example 2 Identifies the current date with datepart:

```
1> select datepart(month, current_date())
2> go

-----------
8

(1 row affected)
```

Usage Finds the current date as it exists on the server.

Standards ANSI SQL – Compliance level: Entry-level compliant.

Permissions Any user can execute current_date.

See also **Datatypes** Date and time datatypes

Commands select, where clause

Functions dateadd, datename, datepart, getdate

current_time

Description

Returns the current time.

Syntax

current_time()

Parameters

None.

Examples

Example 1 Finds the current time:

```
1> select current_date()
2> go

--------------------------
Aug 29 2003

(1 row affected)
```

Example 2 Use with datename:

```
1> select datename(minute, current_time())
2> go

------------------------------
45

(1 row affected)
```

Usage

Finds the current time as it exists on the server

Standards

ANSI SQL – Compliance level: Entry-level compliant.

Permissions

Any user can execute current_time.

See also

Datatypes Date and time datatypes

Commands select, where clause

Functions dateadd, datename, datepart, getdate

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

If a database is open, curunreservedpgs takes the value from memory. If it is not in use, the value is taken from the third parameter you specify in curunreservedpgs. In this example, the value comes from the unreservedpgs column in the sysusages table.

```
select
(dbid), d.name,
    curunreservedpgs(dbid, lstart, unreservedpgs)
    from sysusages u, sysdevices d
where u.vdevno=d.vdevno
and d.status &2 = 2
```

```
                                name
 ------------------------------ ------------------------- -----------
master                          master                           1634
tempdb                          master                            423
model                           master                            423
pubs2                           master                             72
sybsystemdb                     master                            399
sybsystemprocs                  master                           6577
sybsyntax                       master                            359

(7 rows affected)
```

Example 2 Displays the number of free pages on the segment for dbid starting on sysusages.lstart:

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

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 77.
- If a database is open, the value returned by curunreservedpgs is taken from memory. If it is not in use, the value is taken from the third parameter you specify in curunreservedpgs.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute curunreservedpgs.

See also

Functions db_id, lct_admin

data_pages

Description

Returns the number of pages used by the specified table, index, or a specific partition. The result does not include pages used for internal structures.

This function replaces data_pgs and ptn_data_pgs from versions of Adaptive Server earlier than 15.0.

Syntax

data_pages(*dbid*, *object_id* [, *indid* [, *ptnid*]])

Parameters

dbid
is the database ID of the database that contains the data pages.

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

indid
is the index ID of the target index.

ptnid
is the partition ID of the target partition.

Examples

Example 1 Returns the number of pages used by the object with a object ID of 31000114 in the specified database (including any indexes):

```
select data_pages(5, 31000114)
```

Example 2 Returns the number of pages used by the object in the data layer, regardless of whether or not a clustered index exists:

```
select data_pages(5, 31000114, 0)
```

Example 3 Returns the number of pages used by the object in the index layer for a clustered index. This does not include the pages used by the data layer:

```
select data_pages(5, 31000114, 1)
```

Example 4 Returns the number of pages used by the object in the data layer of the specific partition, which in this case is 2323242432:

```
select data_pages(5, 31000114, 0, 2323242432)
```

Usage

In the case of an APL (all-pages lock) table, if a clustered index exists on the table, then passing in an *indid* of:

- 0 – reports the data pages.
- 1 – reports the index pages.

All erroneous conditions return a value of zero, such as when the *object_id* does not exist in the current database, or the targeted *indid* or *ptnid* cannot be found.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute data_pages.

See also **Functions** object_id, row_count

System procedure sp_spaceused

datachange

Description

Measures the amount of change in the data distribution since update statistics last ran. Specifically, it measures the number of inserts, updates, and deletes that have occurred on the given object, partition, or column, and helps you determine if invoking update statistics would benefit the query plan.

Syntax

datachange(*object_name*, *partition_name*, *column_name*)

Parameters

object_name
is the object name in the current database.

partition_name
is the data partition name. This value can be null.

column_name
is the column name for which the datachange is requested. This value can be null.

Examples

Example 1 Provides the percentage change in the au_id column in the author_ptn partition:

```
select datachange("authors", "author_ptn", "au_id")
```

Example 2 Provides the percentage change in the authors table on the au_ptn partition. The null value for the *column_name* parameter indicates that this checks all columns that have historgram statistics and obtains the maximum datachange value from among them.

```
select datachange("authors", "au_ptn", null)
```

Usage

- The datachange function requires all three parameters.
- datachange is a measure of the inserts, deletes and updates but it does not count them individually. datachange counts an update as a delete and an insert, so each update contributes a count of 2 towards the datachange counter.
- The datachange built-in returns the datachange count as a percent of the number of rows, but it bases this percentage on the number of rows remaining, not the original number of rows. For example, if a table has five rows and one row is deleted, datachange reports a value of 25 % since the current row count is 4 and the datachange counter is 1.
- datachange is expressed as a percentage of the total number of rows in the table, or partition if you specify a partition. The percentage value can be greater than 100 percent because the number of changes to an object can be much greater than the number of rows in the table, particularly when the number of deletes and updates happening to a table is very high.

- The value that datachange displays is the in-memory value. This can differ from the on-disk value because the on-disk value gets updated by the housekeeper, when you run sp_flushstats, or when an object descriptor gets flushed.
- The datachange values is not reset when histograms are created for global indexes on partitioned tables.

datachange is reset or initialized to zero when:

- New columns are added, and their datachange value is initialized.
- New partitions are added, and their datachange value is initialized.
- Data-partition-specific histograms are created, deleted or updated. When this occurs, the datachange value of the histograms is reset for the corresponding column and partition.
- Data is truncated for a table or partition, and its datachange value is reset
- A table is repartitioned either directly or indirectly as a result of some other command, and the datachange value is reset for all the table's partitions and columns.
- A table is unpartitioned, and the datachange value is reset for all columns for the table.

datachange has the following restrictions:

- datachange statistics are not maintained on tables in system tempdbs, user-defined tempdbs, system tables, or proxy tables.
- datachange updates are non-transactional. If you roll back a transaction, the datachange values are not rolled back, and these values can become inaccurate.
- If memory allocation for column-level counters fails, Adaptive Server tracks partition-level datachange values instead of column-level values.
- If Adaptive Server does not maintain column-level datachange values, it then resets the partition-level datachange values whenever the datachange values for a column are reset.

Permissions

Any user can execute datachange.

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. *expression* can be of any datatype, an is usually a column name. If *expression* is a character constant, it must be enclosed in quotes.

Examples

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

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

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

Usage

- datalength, a system function, returns the length of *expression* in bytes.
- For columns defined for the Unicode datatype, datalength returns the actual number of bytes of the data stored in each row. For example, this is what is returned if a unitext column ut contains row value U+0041U+0042U+d800dc00:

  ```
  select datalength(ut) from unitable
  -------------
  8
  ```

- datalength finds the actual length of the data stored in each row. datalength is useful on varchar, univarchar, 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 thr defined length.
- datalength of any NULL data returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute datalength.

See also

Functions char_length, col_length

dateadd

Description

Adds an interval to a specified date or time. It takes three arguments: the datepart, a number, and a chronological expression. The datatype of the result is the same as the datatype of the last argument with a value equal to the original value plus the number of date parts. If the last argument is a bigtime, and the datepart is a year, month, or day, the result is the original bigtime argument.

Syntax

dateadd(*date_part*, *integer*, {*date* | *time* | *bigtime* | *datetime*, | *bigdatetime*})

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 73.

numeric
is an integer expression.

date expression
is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

Examples

Example 1 Adds one million microseconds to a bigtime:

```
declare @a bigtime
select @a = "14:20:00.010101"
select dateadd(us, 1000000, @a)
------------------------------
2:20:01.010101PM
```

Example 2 Adds 25 hours to a bigdatetime and the day will increment:

```
declare @a bigdatetime
select @a = "apr 12, 0001 14:20:00 "
select dateadd(hh, 25, @a)
------------------------------
Apr 13 0001   2:20PM
```

Example 3 Displays the new publication dates when the publication dates of all the books in the titles table slip by 21 days:

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

Example 4 Adds one day to a date:

```
declare @a date
select @a = "apr 12, 9999"
select dateadd(dd, 1, @a)
--------------------------
```

```
Apr 13 9999
```

Example 5 Subtracts five minutes to a time:

```
select dateadd(mi, -5, convert(time, "14:20:00"))
--------------------------
2:15PM
```

Example 6 Adds one day to a time and the time remains the same:

```
declare @a time
select @a = "14:20:00"
select dateadd(dd, 1, @a)
--------------------------
2:20PM
```

Example 7 Adds higher values resulting in the values rolling over to the next significant field, even though there are limits for each date_part, as with datetime values:

```
--Add 24 hours to a datetime
select dateadd(hh, 24, "4/1/1979")
--------------------------
Apr  2 1979 12:00AM

--Add 24 hours to a date
select dateadd(hh, 24, "4/1/1979")
-------------------------
Apr  2 1979
```

Usage

- dateadd, a date function, adds an interval to a specified date. For more information about date functions, see "Date functions" on page 73.
- 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 such an addition is meaningful only if the result date returned by dateadd changes by at least one minute.
- If a string is given as an argument in place of the chronological value the server interprets it as a datetime value regardless of its apparent precision. This default behavior may be changed by setting the configuration parameter builtin date strings or the set option builtin_date_strings. When these options are set the server will interpret strings given to chronological builtins as bigdatetimes. See the *System Administration Guide* for more information.

- When a datepart of microseconds is given to this builtin string values will always be interpreted as bigdatetime.
- Use the datetime datatype only for dates after January 1, 1753. datetime values must be enclosed in single or double quotes. Use the date datatype for dates from January 1, 0001 to 9999. date 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 44 and "Datatype conversion functions" on page 63.

  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.

Table 2-8: date_part recognized abbreviations

Date part	Abbreviation	Values
Year	yy	1753 – 9999 (datetime) 1900 – 2079 (smalldatetime) 0001 – 9999 (date)
Quarter	qq	1 – 4
Month	mm	1 – 12
Week	wk	1054
Day	dd	1 – 7
dayofyear	dy	1 – 366
Weekday	dw	1 – 7
Hour	hh	0 – 23
Minute	mi	0 – 59
Second	ss	0 – 59
millisecond	ms	0 – 999
microsecond	us	0 – 999999

Standards

ANSI SQL – Compliance 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

Calculates the number of date parts between two specified dates or times. It takes three arguments. The first is a datepart. The second and third are chronological values. For dates, times, datetimes and bigdatetimes, the result is a signed integer value equal to date2 and date1, in date parts.

•If the second or third argument is a date, and the datepart is an hour, minute, second, millisecond, or microsecond, the dates are treated as midnight.

•If the second or third argument is a time, and the datepart is a year, month, or day, then zero is returned.

•datediff results are truncated, not rounded when the result is not an even multiple of the datepart.

•For the smaller time units, there are overflow values and the function returns an overflow error if you exceed these limits.

Syntax

datediff(*datepart*, {*date, date* | *time*, *time* | *bigtime*, *bigtime* | *datetime*, *datetime* | *bigdatetime*, *bigdatetime*}])

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 73.

date expression1

is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

date expression2

is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

Examples

Example 1 Returns the number of microseconds between two bigdatetimes:

```
declare @a bigdatetime
declare @b bigdatetime
select @a = "apr 1, 1999  00:00:00.000000"
select @b = "apr 2, 1999  00:00:00.000000"
select datediff(us, @a, @b)
--------------------------
86400000000
```

Example 2 Returns the overflow size of milliseconds return value:

```
select datediff(ms, convert(bigdatetime, "4/1/1753"),
convert(bigdatetime, "4/1/9999"))
Msg 535, Level 16, State 0:
Line 2:
```

```
Difference of two datetime fields caused overflow at
runtime.
Command has been aborted
```

Example 3 Finds the number of days that have elapsed between pubdate and the current date (obtained with the getdate function):

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

Example 4 Finds the number of hours between two times:

```
declare @a time
declare @b time
select @a = "20:43:22"
select @b = "10:43:22"
select datediff(hh, @a, @b)
-----------
   -10
```

Example 5 Finds the number of hours between two dates:

```
declare @a date
declare @b date
select @a = "apr 1, 1999"
select @b = "apr 2, 1999"
select datediff(hh, @a, @b)
-----------
     24
```

Example 6 Finds the number of days between two times:

```
declare @a time
declare @b time
select @a = "20:43:22"
select @b = "10:43:22"
select datediff(dd, @a, @b)
-----------
     0
```

Example 7 Returns the overflow size of milliseconds return value:

```
select datediff(ms, convert(date, "4/1/1753"), convert(date, "4/1/9999"))
Msg 535, Level 16, State 0:
Line 2:
Difference of two datetime fields caused overflow at runtime.
Command has been aborted
```

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 73.

- 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 see 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 you use smalldatetime values, 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.
- If the second or third argument is a date, and the datepart is hour, minute, second, or millisecond, the dates are treated as midnight.
- If the second or third argument is a time, and the datepart is year, month, or day, then 0 is returned.
- datediff results are truncated, not rounded, when the result is not an even multiple of the date part.

- If a string is given as an argument in place of the chronological value the server interprets it as a datetime value regardless of its apparent precision. This default behavior may be changed by setting the configuration parameter builtin date strings or the set option builtin_date_strings. When these options are set the server will interpret strings given to chronological builtins as bigdatetimes. See the *System Administration Guide* for more information.
- When a datepart of microseconds is given to this builtin string values will always be interpreted as bigdatetime.
- For the smaller time units, there are overflow values, and the function returns an overflow error if you exceed these limits:
  - Microseconds:approx 3 days
  - Milliseconds: approx 24 days
  - Seconds: approx 68 years
  - Minutes: approx 4083 years
  - Others: No overflow limit

Standards

ANSI SQL – Compliance 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 specified datepart (the first argument) of the specified date or time (the second argument) as a character string. Takes a date, time, bigdatetime, bigtime, datetime, or smalldatetime value as its second argument.

Syntax

datename(*datepart* {*date* | *time* | *bigtime* | *datetime* | *bigdatetime*})

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 73.

date_expression
is an expression of type datetime, smalldatetime, bigdatetime, bigtime, time or a character string in a datetime format.

Examples

Example 1 Finds the month name of a bigdatetime:

```
declare @a bigdatetime
select @a = "apr 12, 0001 00:00:00.010101"
select datename(mm, @a)
------------------------------
April
```

Example 2 Assumes a current date of November 20, 2000:

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

November
```

Example 3 Finds the month name of a date:

```
declare @a date
select @a = "apr 12, 0001"
select datename(mm, @a)
------------------------------
    April
```

Example 4 Finds the seconds of a time:

```
declare @a time
select @a = "20:43:22"
select datename(ss, @a)
------------------------------
    22
```

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 73.
- 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

ANSI SQL – Compliance 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 specified datepart in the first argument of the specified date, and the second argument, as an integer. Takes a date, time, datetime, bigdatetime, bigtime, or smalldatetime value as its second argument. If the datepart is hour, minute, second, millisecond, or microsecond, the result is 0.

Syntax

datepart(*date_part* {*date* | *time* | *datetime* | *bigtime* | *bigdatetime*}))

Parameters

date_part

is a date part. Table 2-9 lists the date parts, the abbreviations recognized by datepart, and the acceptable values.

Table 2-9: Date parts and their values

Date part	Abbreviation	Values
year	yy	1753 – 9999 (2079 for smalldatetime). 0001 to 9999 for date
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
microsecond	us	0 - 999999
calweekofyear	cwk	1 – 53
calyearofweek	cyr	1753 – 9999 (2079 for smalldatetime). 0001 to 9999 for date
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.

  For datetime, smalldatetime, and time types 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.

  Microseconds must be preceded by a decimal point and represent fractions of a second.

date_expression
is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

Examples

Example 1 Finds the microseconds of a bigdatetime:

```
declare @a bigdatetime
select @a = "apr 12, 0001 12:00:00.000001"
select datepart(us, @a)
-----------
000001
```

Example 2 Assumes a current date of November 25, 1995:

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

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

Example 3 Returns the year of publication from traditional cookbooks:

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

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

Example 4

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

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

Example 5

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

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

Example 6

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

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

Example 7 Find the hours in a time:

```
declare @a time
select @a = "20:43:22"
select datepart(hh, @a)
-----------
    20
```

Example 8 Returns 0 (zero) if an hour, minute, or second portion is requested from a date using datename or datepar) the result is the default time; Returns the default date of Jan 1 1990 if month, day, or year is requested from a time using datename or datepart:

```
--Find the hours in a date
declare @a date
select @a = "apr 12, 0001"
select datepart(hh, @a)
-----------
    0

--Find the month of a time
declare @a time
select @a = "20:43:22"
select datename(mm, @a)
------------------------------
January
```

When you give a null value to a datetime function as a parameter, NULL is returned.

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 73.

- 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 U.S. English as the default language, the following returns 1988:

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

  However, the following returns 1989:

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

  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 U.S. English as their default language, the first day of the week is 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.You can change the default behavior on a per-session basis with set datefirst. See the datefirst option of the set command for more information.

- 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 datetime and time are only accurate to 1/300th of a second, when these datatypes are used with datepart, milliseconds are rounded to the nearest 1/300th second.

- 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 — ANSI SQL – Compliance 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

day

Description

Returns an integer that represents the day in the datepart of a specified date.

Syntax

day(*date_expression*)

Parameters

date_expression
is an expression of type datetime, smalldatetime, date, or a character string in a datetime format.

Examples

Returns the integer 02:

```
day("11/02/03")
----------
02
```

Usage

day(*date_expression*) is equivalent to datepart(dd,*date_expression*).

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute day.

See also

Datatypes datetime, smalldatetime, date, time

Functions datepart, month, year

db_attr

Description

Returns the durability, dml_logging, and template settings for the specified database.

Syntax

db_attr('*database_name*' | *database_ID* | NULL, 'attribute')

Parameters

database_name
name of the database.

database_ID
ID of the database

NULL
if included, db_attr reports on the current database

attribute
is one of:

- help – display db_attr usage information.
- durability – returns durability of the given database. full, at_shutdown, or no_recovery.
- dml_logging – returns the value for DML logging for specified database: full or minimal.
- template – returns the name of the template database used for the specified database. If no database was used as a template to create the database, returns NULL.

Examples

Example 1 Returns the syntax for db_attr:

```
select db_attr(0, "help")
Usage: db_attr('dbname' | dbid | NULL, 'attribute')
List of options in attributes table:
         0 : help
         1 : durability
         2 : dml_logging
         3 : template
-----------------------------------------------------
NULL

select abs(-1)

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

Example 2 Selects the name, durability setting, dml_logging setting and template used from sysdatabses:

```
select  name = convert(char(20), name),
durability = convert(char(15), db_attr(name,     "durability")),
     dml_logging = convert(char(15), db_attr(dbid,     "dml_logging")),
     template = convert(char(15), db_attr(dbid, "template"))
from sysdatabases
name              durability         dml_logging                 template
----------------   ----------------   ------------------------    --------
master             full               full                        NULL
model              full               full                        NULL
tempdb             no_recovery        full                        NULL
sybsystemdb        full               full                        NULL
sybsystemprocs     full               full                        NULL
repro              full               full                        NULL
imdb               no_recovery        full                        db1
db                 full               full                        NULL
at_shutdown_db     at_shutdown        full                        NULL
db1                full               full                        NULL
dml                at_shutdown        minimal                     NULL
```

Example 3 Runs db_attr against the DoesNotExist database, which does not exist:

```
select db_attr("DoesNotExist", "durability")
-----------------------------------------------------
NULL
```

Example 4 Runs db_attr against a database with an ID of 12345, which does not exist:

```
select db_attr(12345, "durability")
-----------------------------------------------------
NULL
```

Example 5 Runs db_attr against an attribute that does not exist:

```
select db_attr(1, "Cmd Does Not Exist")
-----------------------------------------------------
NULL
```

Usage .

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute db_attr.

See also

Functions

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 — Returns the ID number of sybsystemprocs:

```
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 77.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute db_id.

See also — **Functions** db_name, object_id

db_instanceid

Description

Cluster environments only – returns the ID of the owning instance of a specified local temporary database. Returns NULL if the specified database is a global temporary database or a nontemporary database.

Syntax

db_instanceid(*database_id*)
db_instanceid(*database_name*)

Parameters

database_id
ID of the database.

database_name
name of the database

Examples

Returns the owning instance for database ID 5

```
select db_instanceid(5)
```

Usage

- Access to a local temporary database is allowed only from the owning instance. db_instanceid determines whether the specified database is a local temporary database, and the owning instance for the local temporary database.You can then connect to the owning instance and access its local temporary database.
- You must include an parameter with db_instanceid.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can run sdc_intempdbconfig.

db_name

Description
Returns the name of the database where the 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 Returns the name of the current database:

```
select db_name()
```

Example 2 Returns the name of database ID 4:

```
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 77.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute db_name.

See also
Functions col_name, db_id, object_name

db_recovery_status

Description

Cluster environments only – returns the recovery status of the specified database. Returns the recovery status of the current database if you do not include a value for *database_ID* or *database_name*.

Syntax

db_recovery_status([*database_ID* | *database_name*])

Parameters

database_ID
is the ID of the database whose recovery status you are requesting.

database_name
is the name of the database whose recovery status you are requesting.

Examples

Example 1 Returns the recovery status of the current database:

```
select db_recovery_status()
```

Example 2 Returns the recovery status of the database with named test:

```
select db_recovery_status("test")
```

Example 3 Returns the recovery status of a database with a database id of 8:

```
select db_recovery_status(8)
```

Usage

A return value of 0 indicates the database is not in node-failover recovery. A return value of 1 indicates the database is in node-failover recovery.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute db_recovery_status.

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.

- For general information about mathematical functions, see "Mathematical functions" on page 74.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute degrees.

See also
Function radians

derived_stat

Description

Returns derived statistics for the specified object and index.

Syntax

```
derived_stat("object_name" | object_id,
        index_name | index_id,
        ["partition_name" | partition_id,]
        “statistic”)
```

Parameters

object_name

is the name of the object you are interested in. If you do not specify a fully qualified object name, derived_stat searches the current database.

object_id

is an alternative to *object_name*, and is the object ID of the object you are interested in. *object_id* must be in the current database

index_name

is the name of the index, belonging to the specified object that you are interested in.

index_id

is an alternative to *index_name*, and is the index ID of the specified object that you are interested in.

partition_name

is the name of the partition, belonging to the specific partition that you are interested in. *partition_name* is optional. When you use *partition_name* or *partition_id*, Adaptive Server returns statistics for the target partition, instead of for the entire object.

partition_id

is an alternative to *partition_name*, and is the partition ID of the specified object that you are interested in. *partition_id* is optional.

"*statistic*"

the derived statistic to be returned. Available statistics are:

- data page cluster ratio or dpcr – the data page cluster ratio for the object/index pair
- index page cluster ratio or ipcr – the index page cluster ratio for the object/index pair
- data row cluster ratio or drcr – the data row cluster ratio for the object/index pair
- large io efficiency or lgio – the large I/O efficiency for the object/index pair
- space utilization or sput – the space utilization for the object/index pair

Examples

Example 1 Selects the space utilization for the titleidind index of the titles table:

```
select derived_stat("titles", "titleidind", "space utilization")
```

Example 2 Selects the data page cluster ratio for index ID 2 of the titles table. Note that you can use either `"dpcr"` or `"data page cluster ratio"`:

```
select derived_stat("titles", 2, "dpcr")
```

Example 3 Statistics are reported for the entire object, as neither the partition ID nor name is not specified:

```
1> select derived_stat(object_id("t1"), 2, "drcr")
2> go

---------------------------
                   0.576923
```

Example 4 Reports the statistic for the partition tl_928003396:

```
1> select derived_stat(object_id("t1"), 0, "t1_928003306", "drcr")
2> go

 ---------------------------
                    1.000000

(1 row affected)
```

Example 5 Selects derived statistics for all indexes of a given table, using data from syspartitions:

```
select convert(varchar(30), name) as name, indid,
    convert(decimal(5, 3), derived_stat(id, indid, 'sput')) as 'sput',
    convert(decimal(5, 3), derived_stat(id, indid, 'dpcr')) as 'dpcr',
    convert(decimal(5, 3), derived_stat(id, indid, 'drcr')) as 'drcr',
```

```
    convert(decimal(5, 3), derived_stat(id, indid, 'lgio')) as 'lgio'
from syspartitions where id = object_id('titles')
go

 name                           indid  sput     dpcr     drcr     lgio
 ------------------------------ ------ -------- -------- -------- --------
 titleidind_2133579608               1    0.895    1.000    1.000    1.000
 titleind_2133579608                 2    0.000    1.000    0.688    1.000

(2 rows affected)
```

Example 6 Selects derived statistics for all indexes and partitions of a partitioned table. Here, mymsgs_rr4 is a roundrobin partitioned table that is created with a global index and a local index.

```
1> select * into mymsgs_rr4 partition by roundrobin 4 lock datarows
2> from master..sysmessages
2> go

(7597 rows affected)

1> create clustered index mymsgs_rr4_clustind on mymsgs_rr4(error, severity)
2> go
1> create index mymsgs_rr4_ncind1 on mymsgs_rr4(severity)
2> go
1> create index mymsgs_rr4_ncind2 on mymsgs_rr4(langid, dlevel) local index
2> go

2> update statistics mymsgs_rr4
1>

2> select convert(varchar(10), object_name(id)) as name,
3>      (select convert(varchar(20), i.name) from sysindexes i
4>       where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where id = object_id('mymsgs_rr4')
 name                        ptnname                    indid sput dpcr drcr lgio
---------------------------- -------------------------- ----- ---- ---- ---- ----
mymsgs_rr4 mymsgs_rr4          mymsgs_rr4_786098810          0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4          mymsgs_rr4_802098867          0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4          mymsgs_rr4_818098924          0 0.89 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4          mymsgs_rr4_834098981          0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_clustind mymsgs_rr4_clustind_850099038 2 0.83 0.995 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind1   mymsgs_rr4_ncind1_882099152   3 0.99 0.445 0.88 1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_898099209   4 0.15 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_914099266   4 0.88 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_930099323   4 0.877 1.000 1.000
```

```
1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_946099380   4 0.945 0.993 1.000
1.000
```

Example 7 Selects derived statistics for all allpages-locked tables in the current database:

```
2> select convert(varchar(10), object_name(id)) as name
3>     (select convert(varchar(20), i.name) from sysindexes i
4>      where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where lockscheme(id) = "allpages"
12>   and (select o.type from sysobjects o where o.id = p.id) = 'U'

 name                            ptnname                    indid sput dpcr drcr lgio
----------- ----------------- --------------------------- ----- ---- ---- ---- ----
 stores     stores            stores_18096074               0 0.276 1.000 1.000 1.000
 discounts  discounts         discounts_50096188            0 0.075 1.000 1.000 1.000
 au_pix     au_pix            au_pix_82096302               0 0.000 1.000 1.000 1.000
 au_pix     tau_pix           tau_pix_82096302             255 NULL NULL NULL NULL
 blurbs     blurbs            blurbs_114096416              0 0.055 1.000 1.000 1.000
 blurbs     tblurbs           tblurbs_114096416            255 NULL NULL NULL NULL
 t1apl      t1apl             t1apl_1497053338              0 0.095 1.000 1.000 1.000
 t1apl      t1apl             t1apl_1513053395              0 0.082 1.000 1.000 1.000
 t1apl      t1apl             t1apl_1529053452              0 0.095 1.000 1.000 1.000
 t1apl      t1apl_ncind       t1apl_ncind_1545053509        2 0.149 0.000 1.000 1.000
 t1apl      t1apl_ncind_local t1apl_ncind_local_1561053566 3 0.066 0.000 1.000 1.000
 t1apl      t1apl_ncind_local t1apl_ncind_local_1577053623 3 0.057 0.000 1.000 1.000
 t1apl      t1apl_ncind_local t1apl_ncind_local_1593053680 3 0.066 0.000 1.000 1.000
 authors    auidind           auidind_1941578924            1 0.966 0.000 1.000 1.000
 authors    aunmind           aunmind_1941578924            2 0.303 0.000 1.000 1.000
 publishers pubind            pubind_1973579038             1 0.059 0.000 1.000 1.000
 roysched   roysched          roysched_2005579152           0 0.324 1.000 1.000 1.000
 roysched   titleidind        titleidind_2005579152         2 0.777 1.000 0.941 1.000
 sales      salesind          salesind_2037579266           1 0.444 0.000 1.000 1.000
 salesdetai salesdetail       salesdetail_2069579380        0 0.614 1.000 1.000 1.000
 salesdetai titleidind        titleidind_2069579380         2 0.518 1.000 0.752 1.000
 salesdetai salesdetailind    salesdetailind_2069579380     3 0.794 1.000 0.726 1.000
 titleautho taind             taind_2101579494              1 0.397 0.000 1.000 1.000
 titleautho auidind           auidind_2101579494            2 0.285 0.000 1.000 1.000
 titleautho titleidind        titleidind_2101579494         3 0.223 0.000 1.000 1.000
 titles     titleidind        titleidind_2133579608         1 0.895 1.000 1.000 1.000
 titles     titleind          titleind_2133579608           2 0.402 1.000 0.688 1.000

(27 rows affected)
```

Usage

- derived_stat returns a double precision value.
- The values returned by derived_stat match the values presented by the optdiag utility.
- If the specified object or index does not exist, derived_stat returns NULL.
- Specifying an invalid statistic type results in an error message.
- Using the optional *partition_name* or *partition_id* reports the requested statistic for the target partition; otherwise, derived_stat reports the statistic for the entire object.
- If you provide:
  - Four arguments – derived_stat uses the third argument as the partition, and returns derived statistics on the fourth argument.
  - Three arguments – derived_stat assumes you did not specifiy a partition, and returns derived statistic specified by the third argument.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Only the table owner can execute derived_stat.

See also

Document *Performance and Tuning Guide* for:

- “Access Methods and Query Costing for Single Tables”
- “Statistics Tables and Displaying Statistics with optdiag”

Utility optdiag

difference

Description Returns the difference between two soundex values.

Syntax difference(*expr1*,*expr2*)

Parameters

expr1
is a character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or unichar type.

expr2
is another character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or 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 *expr1* or *expr2* is NULL, returns NULL.
- If you give 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 76.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute difference.

See also **Function** soundex

exp

Description
Returns the value that results from raising the constant 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 74.

Standards
ANSI SQL – Compliance 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, int, or bigint), 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 74.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute floor.

See also

Functions abs, ceiling, round, sign

get_appcontext

Description

Returns the value of the attribute in a specified context. get_appcontext is a built-in function provided by the Application Context Facility (ACF).

Syntax

get_appcontext (“*context_name*”, “*attribute_name*”)

Parameters

context_name
is a row specifying an application context name, saved as datatype char(30).

attribute_name
is a row specifying an application context attribute name, saved as char(30).

Examples

Example 1 Shows VALUE1 returned for ATTR1.

```
select get_appcontext("CONTEXT1", "ATTR1")

--------------
VALUE1
```

ATTR1 does not exist in CONTEXT2:

```
select get_appcontext("CONTEXT2", "ATTR1")
```

Example 2 Shows the result when a user without appropriate permissions attempts to get the application context.

```
select get_appcontext("CONTEXT1", "ATTR2", "VALUE1")

Select permission denied on built-in get_appcontext, database dbid
-----------
-1
```

Usage

- This function returns 0 for success and -1 for failure.
- If the attribute you require does not exist in the application context, get_appcontext returns NULL.
- get_appcontext saves attributes as char datatypes. If you are creating an access rule that compares the attribute value to other datatypes, the rule should convert the char data to the appropriate datatype.
- All arguments for this function are required.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Permissions depend on the user profile and the application profile, and are stored by the ACF.

See also

For more information on the ACF, see “Row-level access control” in Chapter 11, “Managing User Permissions” of the *System Administration Guide*.

Functions get_appcontext, list_appcontext, rm_appcontext, set_appcontext

getdate

Description
Returns the current system date and time.

Syntax
getdate()

Parameters
None.

Examples

Example 1 Assumes a current date of November 25, 1995, 10:32 a.m.:

```
select getdate()
```

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

Example 2 Assumes a current date of November:

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

```
11
```

Example 3 Assumes a current date of November:

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

```
November
```

Usage

- getdate, a date function, returns the current system date and time.
- For more information about date functions, see “Date functions” on page 73.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute getdate.

See also
Datatypes Date and time datatypes

Functions dateadd, datediff, datename, datepart

getutcdate

Description

Returns a date and time where the value is in Universal Coordinated Time (UTC). getutcdate is calculated each time a row is inserted or selected.

Syntax

getutcdate()

Examples

```
insert t1 (c1, c2, c3) select c1, getutcdate(),
getdate() from t2)
```

See also

Functions biginttohex, convert

has_role

Description

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

Syntax

has_role ("*role_name*", *option*)

Parameters

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

option
allows you to limit the scope of the information returned. Currently, the only option supported is 1, which suppresses auditing.

Examples

Example 1 Creates a procedure to check if the user is a System Administrator:

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

Example 2 Checks that the user has been granted the System Security Officer role:

```
select has_role("sso_role", 1)
```

Example 3 Checks that the user has been granted the Operator role:

```
select has_role("oper_role", 1)
```

Usage

- has_role functions the same way proc_role does. Beginning with Adaptive Server version 15.0, Sybase supports—and recommends—that you use has_role instead of proc_role. You need not, hoever, convert all of your existing uses of proc_role to has_role.
- has_role, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- has_role returns 0 if the user has:
    - Not been granted the specified role
    - Not been granted a role which contains the specified role
    - Been granted, but has not activated, the specified role
- has_role returns 1 if the invoking user has been granted, and has activated, the specified role.
- has_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 77.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute has_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

hash

Description

Produces a fixed-length hash value expression.

Syntax

hash(*expression* , [*algorithm*])

Parameters

expression

is the value to be hashed. This can be a column name, variable, constant expression, or any combination of these that evaluates to a single value. It cannot be image, text, unitext, or off-row Java datatypes. Expression is usually a column name. If expression is a character constant, it must be enclosed in quotes.

algorithm

is the algorithm used to produce the hash value. A character literal (not a variable or column name) that can take the values of either md5 or sha1, 2 (meaning md5 binary), or 3 (meaning sha1 binary). If omitted, md5 is used.

Algorithm	Results in
hash(*expression*, 'md5')	A varchar 32-byte string. md5 (Message Digest Algorithm 5) is the cryptographic hash function with a 128-bit hash value.
hash(*expression*)	A varchar 32-byte string
hash(*expression*, 'sha1')	A varchar 40-byte string sha1 (Secure Hash Algorithm) is the cryptographic hash function with a 160-bit hash value.
hash(*expression*, 2)	A varbinary 16-byte value (using the md5 algorithm)
hash(*expression*, 3)	A varbinary 20-byte value (using the sha1 algorithm)

Examples

This example shows how a seal is implemented. The existence of a table called "atable" and with columns id, sensitive_field and tamper seal.

```
update atable set tamper_seal=hash(convert(varchar(30),
id) + sensitive_field+@salt, 'sha1')
```

Usage

When specified as a character literal, *algorithm* is not case-sensitive—"md5", "Md5" and "MD5" are equivalent. However, if *expression* is specified as a character datatype then the value is case sensitive. "Time," "TIME," and "time" will produce different hash values.

If *algorithm* is a character literal, the result is a varchar string. For "md5" this is a 32-byte string containing the hexadecimal representation of the 128-bit result of the hash calculation. For "sha1" this is a 40-byte string containing the hexadecimal representation of the 160-bit result of the hash calculation.

If *algorithm* is an integer literal, the result is a varbinary value. For 2, this is a 16-byte value containing the 128-bit result of the hash calculation. For 3, this is a 20-byte value containing the 160-bit result of the hash calculation.

Note Trailing null values are trimmed by Adaptive Server when inserted into varbinary columns.

Individual bytes that form *expression* are fed into the hash algorithm in the order they appear in memory. For many datatypes order is significant. For example, the binary representation of the 4-byte INT value 1 will be 0x00, 0x00, 0x00, 0x01 on MSB-first (big-endian) platforms and 0x01, 0x00, 0x00, 0x00 on LSB-first (little-endian) platforms. Because the stream of bytes is different between platforms, the hash value is different as well. Use hashbytes function to achieve platform independent hash value.

Note The hash algorithms MD5 and SHA1 are no longer considered entirely secure by the cryptographic community. As for any such algorithm, you should be aware of the risks of using MD5 or SHA1 in a security-critical context.

Standards — SQL92- and SQL99- compliant

Permissions — Any user can execute hash.

See also — See also hashbytes for platform independent hash values.

hashbytes

Description
Produces a fixed-length, hash value expression.

Syntax
hashbytes(*algorithm*, *expression*[, *expression...*] [, using *options*])

Parameters
expression[, *expression*...]
– is the value to be hashed. This value can be a column name, variable, constant expression, or a combination of these that produces a single value. It cannot be image, text, unitext, or off-row Java datatypes.

algorithm
is the algorithm used to produce the hash value. A character literal (not a variable or a column name) that can take the values "md5", "sha", "sha1", or "ptn".

Algorithm	Description
Md5	Message Digest Algorithm 5 – is the cryptographic hash algorithm with a 128 bit hash value. hashbytes('md5', *expression*[,...]) results in a varbinary 16-byte value.
Sha-Sha1	Secure Hash Algorithm – is the cryptographic hash algorithm with a 160-bit hash value. hashbytes('shal', *expression*[,...]) results in a varbinary 20-byte value.
Ptn	The partition hash algorithm with 32-bit hash value. The *using* clause is ignored for the 'ptn' algorithm. hashbytes(*'ptn'*, *expression*[,...]) results in an unsigned int 4-byte value.
using	Orders bytes for platform independence. The optional using clause can precede the following option strings: • lsb – all byte-order dependent data is normalized to little-endian byte-order before being hashed. • msb – all byte-order dependent data is normalized to big-endian byte-order before being hashed. • unicode – character data is normalized to unicode (UTF–16) before being hashed. **Note** A UTF – 16 string is similar to an array of short integers. Because it is byte-order dependent, Sybase suggest for platform independence you use lsb or msb in conjunction with UNICODE. • unicode_lsb – a combination of unicode and lsb. • unicode_msb – a combination of unicode and msb.

Examples
Example 1 Seals each row of a table against tampering. This example assumes the existence of a user table called "xtable" and col1, col2, col3 and tamper_seal.

```
update xtable set tamper_seal=hashbytes('sha1', col1,
col2, col4, @salt)
--
declare @nparts unsigned int
select @nparts= 5
select hashbytes('ptn', col1, col2, col3) % nparts from
xtable
```

Example 2 Shows how col1, col2, and col3 will be used to partition rows into five partitions.

```
alter table xtable partition by hash(col1, col2, col3) 5
```

Usage

The algorithm parameter is not case-sensitive; "md5," "Md5" and "MD5" are all equivalent. However, if the *expression* is specified as a character datatype, the value is case sensitive. "Time," "TIME," and "time" will produce different hash values.

Note Trailing null values are trimmed by Adaptive Server when inserting into varbinary columns.

In the absence of a using clause, the bytes that form *expression* are fed into the hash algorithm in the order they appear in memory. For many datatypes, order is significant. For example, the binary representation of the 4-byte INT value 1 will be 0x00, 0x00, 0x00, 0x01, on MSB-first (big-endian) platforms and 0x01, 0x00, 0x00, 0x00 on LSB-first (little-endian) platforms. Because the stream of bytes is different for different platforms, the hash value is different as well.

With the using clause, the bytes that form *expression* can be fed into the hashing algorithm in a platform-independent manner. The using clause can also be used to transform character data into Unicode so that the hash value becomes independent of the server's character configuration.

Note The hash algorithms MD5 and SHA1 are no longer considered entirely secure by the cryptographic community. Be aware of the risks of using MD5 or SHA1 in a security-critical context.

Standards

SQL92- and SQL99-compliant

Permissions

Any user can execute hashbyte.

See also

See also hash for platform dependent hash values.

hextobigint

Description

Returns the bigint value equivalent of a hexadecimal string

Syntax

hextobigint(*hexadecimal_string*)

Parameters

hexadecimal_string

is the hexadecimal value to be converted to an big integer; must be a character-type column, variable name, or a valid hexadecimal string, with or without a “0x” prefix, enclosed in quotes.

Examples

The following example converts the hexadecimal string 0x7fffffffffffffff to a big integer.

```
1> select hextobigint("0x7fffffffffffffff")
2> go
--------------------
9223372036854775807
```

Usage

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

  hextobigint returns the bigint equivalent of the hexadecimal string. The function always returns the same bigint equivalent for a given hexadecimal string, regardless of the platform on which it is executed.

See also

Functions biginttohex, convert, inttohex, hextoint

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; must be a character-type column, variable name, or a valid hexadecimal string, with or without a "0x" prefix, enclosed in quotes.

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

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

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 63.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute hextoint.

See also
Functions biginttohex, convert, inttohex

host_id

Description

Returns the client computer's operating system process ID for the current Adaptive Server client.

Syntax

host_id()

Parameters

None.

Examples

In this example, the name of the client computer is "ephemeris" and the process ID on the computer "ephemeris" for the Adaptive Server client process is 2309:

```
select host_name(), host_id()
----------------------------- -----------------------
 ephemeris                     2309
```

The following is the process information, gathered using the UNIX ps command, from the computer "ephemeris" showing that the client in this example is "isql" and its process ID is 2309:

```
2309 pts/2    S  0:00 /work/as125/OCS-12_5/bin/isql
```

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 76.

Standards

ANSI SQL – Compliance 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 77.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute host_name.

See also
Function host_id

instance_id

Description
Cluster environments only – Returns the id of the named instance, or the instance from which it is issued if you do not provide a value for *name*.

Syntax
instance_id([*name*])

Parameters
name
name of the instance whose ID you are researching.

Examples
Returns the ID of the local instance:

```
select instance_id()
```

Usage
Returns the ID of the instance named "myserver1":

```
select instance_id(myserver1)
```

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute instance_id.

identity_burn_max

Description

Tracks the identity burn max value for a given table. This function returns only the value; does not perform an update.

Syntax

identity_burn_max(*table_name*)

Parameters

table_name
is the name of the table selected.

Examples

```
select identity_burn_max("t1")

t1
------------
51
```

Usage

identity_burn_max tracks the identity burn max value for a given table.

Permissions

Only the table owner, System Administrator, or database administrator can issue this command.

index_col

Description

Returns the name of the indexed column in the specified table or view, and can be up to 255 bytes in length

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

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

```
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
```

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 76.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute index_col.

See also

Function object_id

System procedure 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 — Returns "DESC" because the salesind index on the sales table is in descending order:

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

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

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 76.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute index_colorder.

index_name

Description

Returns an index name, when you provide the index ID, the database ID, and the object on which the index is defined.

Syntax

index_name(*dbid*, *objid*, *indid*)

Parameters

dbid
is the ID of the database on which the index is defined.

objid
is the ID of the table (in the specified database) on which the index is defined.

indid
is the ID of the index for which you want a name.

Examples

Example 1

Illustrates the normal usage of this function.

```
select index_name(db_id("testdb"),
    object_id("testdb...tab_apl"),1)
------------
```

Example 2 Illustrates the output if the database ID is NULL and you use the current database ID.

```
select index_name(NULL,object_id("testdb..tab_apl"),1)
--------------------------
```

Example 3 Displays the table name if the index ID is 0, and the database ID and object ID are valid.

```
select index_name(db_id("testdb"),
    object_id("testdb..tab_apl"),1)
-----------------------------
```

Usage

- index_name uses the current database ID, if you pass a NULL value in the *dbid* parameter
- index_name returns NULL if you pass a NULL value in the *dbid* parameter.
- index_name returns the object name, if the index ID is 0, and you pass valid inputs for the object ID and the database ID.

Permissions

Any user can execute this function.

See also

db_id, object_id

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 63.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute inttohex.

See also
Functions convert, hextobigint, hextoint

isdate

Description

Determines whether an input expression is a valid datetime value.

Syntax

```
isdate(character_expression)
```

Parameters

character_expression
is a character-type variable, constant expression, or column name.

Examples

Example 1 Determines if the string 12/21/2005 is a valid datetime value:

```
select isdate('12/21/2005')
```

Example 2 Determines if stor_id and date in the sales table are valid datetime values:

```
select isdate(stor_id), isdate(date) from sales
---- ----
0    1
```

store_id is not a valid datetime value, but date is.

Usage

Returns 1 if the expression is a valid datetime value; returns 0 if it is not. Returns 0 for NULL input.

isnumeric

Description

Determines if an expression is a valid numeric datatype.

Syntax

```
isnumeric (character_expression)
```

Parameters

character_expression
is a character-type variable, constant expression, or a column name.

Examples

Example 1 Determines if the values in the postalcode column of the authors table contains valid numeric datatypes:

```
select isnumeric(postalcode) from authors
```

Example 2 Determines if the value $100.12345 is a valid numeric datatype:

```
select isnumeric("$100.12345")
```

Usage

- Returns 1 if the input expression is a valid integer, floating point number, money or decimal type; returns 0 if it does not or if the input is a NULL value. A return value of 1 guarantees that you can convert the expression to one of these numeric types.
- You can include currency symbols as part of the input.

is_quiesced

Description

Indicates whether a database is in quiesce database mode. is_quiesced returns 1 if the database is quiesced and 0 if it is not.

Syntax

is_quiesced(*dbid*)

Parameters

dbid
is the database ID of the database.

Examples

Example 1 Uses the test database, which has a database ID of 4, and which is not quiesced:

```
1> select is_quiesced(4)
2> go

-----------
          0

(1 row affected)
```

Example 2 Uses the test database after running quiesce database to suspend activity:

```
1> quiesce database tst hold test
2> go
1> select is_quiesced(4)
2> go

-----------
          1

(1 row affected)
```

Example 3 Uses the test database after resuming activity using quiesce database:

```
1> quiesce database tst release
2> go
1> select is_quiesced(4)
2> go

 -----------
          0

(1 row affected)
```

Example 4 Executes a select statement with is_quiesced using an invalid database ID:

```
1>select is_quiesced(-5)
```

```
2> go
 -----------
        NULL

(1 row affected)
```

Usage

- is_quiesced has no default values. You see an error if you execute is_quiesced without specifying a database.
- is_quiesced returns NULL if you specify a database ID that does not exist.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute is_quiesced.

See also

Command quiesce database

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, execute:

  ```
  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

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute is_sec_service_on.

See also

Function show_sec_services

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

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

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

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 76.
- The datatypes of the expressions must convert implicitly, or you must use the convert function.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute isnull.

See also

Function convert

isnumeric

Description

Determines if an expression is a valid numeric datatype.

Syntax

isnumeric (*character_expression*)

Parameters

character_expression
is a character-type variable, constant expression, or a column name.

Examples

Example 1 Determines if the values in the postalcode column of the authors table contains valid numeric datatypes:

```
select isnumeric(postalcode) from authors
```

Example 2 Determines if the value $100.12345 is a valid numeric datatype:

```
select isnumeric("$100.12345")
```

Usage

- Returns 1 if the input expression is a valid integer, floating point number, money or decimal type; returns 0 if it does not or if the input is a NULL value. A return value of 1 guarantees that you can convert the expression to one of these numeric types.
- You can include currency symbols as part of the input.

instance_name

Description

Cluster environments only – Returns the name for the Adaptive Server whose id you provide, or the name of the Adaptive Server from which it is issued if you do not provide a value for *id*.

Syntax

instance_name([*id*])

Parameters

id
is the ID of the Adaptive Server whose name you are researching.

Examples

Returns the name of the instance with an ID of 12:

```
select instance_name(12)
```

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute instance_name.

lc_id

Description

Cluster environments only – Returns the ID of the logical cluster whose name you provide, or the current logical cluster if you do not provide a name.

Syntax

lc_id(*logical_cluster_name*)

Parameters

logical_cluster_name
is the name of the logical cluster.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute lc_id

lc_name

Description — Cluster environments only – Returns the name of the logical cluster whose id you provide, or the current logical cluster if you do not provide an ID.

Syntax — lc_name([*logical_cluster_ID*])

Parameters — *logical_cluster_ID*
is the ID of the logical cluster.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute lc_name.

lct_admin

Description

Manages the last-chance threshold, returns the current value of the last-chance threshold (LCT), and aborts transactions in a transaction log that has reached its LCT.

Syntax

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

Parameters

lastchance
creates a LCT in the specified database.

logfull
returns 1 if the LCT has been crossed in the specified database and 0 if it has not.

reserved_for_rollbacks
determines the number of pages a database currently reserved for rollbacks.

database_id
specifies the database.

reserve
obtains either the current value of the LCT 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 LCT.

0
returns the current value of the LCT. The size of the LCT 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 LCT 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.

logsegment_freepages
describes the free space available for the log segment. This is the total value of free space, not per-disk.

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

Examples

Example 1 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:

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

Example 2 Returns 1 if the last-chance threshold for the database with dbid of 6 has been crossed, and 0 if it has not:

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

Example 3 Calculates and returns the number of log pages that would be required to successfully dump the transaction log in a log containing 64 pages:

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

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

Example 4 Returns the current last-chance threshold of the transaction log in the database from which the command was issued:

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

Example 5 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:

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

Example 6 Aborts all open transactions in the database with dbid of 5. This form awakens any processes that may be suspended at the log segment last-chance threshold:

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

Example 7 Determines the number of pages reserved for rollbacks in the pubs2 database, which has a dbid of 5:

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

Example 8 Describes the free space available for a database with a dbid of 4:

```
select lct_admin("logsegment_freepages", 4)
```

Usage

- lct_admin, a system function, manages the log segment's last-chance threshold. For general information about system functions, see "System functions" on page 77.
- 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.

Standards

ANSI SQL – Compliance 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

Document *System Administration Guide*.

Command dump transaction

Function curunreservedpgs

System procedure sp_thresholdaction

left

Description Returns a specified number of characters on the left end of a character string.

Syntax left(*character_expression*, *integer_expression*)

Parameters *character_expression*
is the character string from which the characters on the left are selected.

integer_expression
is the positive integer that specifies the number of characters returned. An error is returned if *integer_expression* is negative.

Examples **Example 1** Returns the five leftmost characters of each book title.

```
use pubs
select left(title, 5) from titles
order by title_id

-----
The B
Cooki
You C
.....
Sushi

(18 row(s) affected)
```

Example 2 Returns the two leftmost characters of the character string “abcdef”:

```
select left("abcdef", 2)
---------
ab
(1 row(s) affected)
```

Usage

- *character_expression* can be of any datatype (except text or image) that can be implicitly converted to varchar or nvarchar. *character_expression* can be a constant, variable, or a column name. You can explicitly convert character_expression using convert.
- left is equivalent to substring(*character_expression*, *1*, *integer_expression*). For more information on this function, see substring on page 297.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute left.

See also **Datatypes** varchar, nvarchar

Functions len, str_replace, substring

len

Description

Returns the number of characters, not the number of bytes, of a specified string expression, excluding trailing blanks.

Syntax

len(*string_expression*)

Parameters

string_expression
is the string expression to be evaluated.

Examples

Returns the characters

```
select len(notes) from titles
where title_id = "PC9999"
-----------
39
```

Usage

This function is the equivalent of char_length(*string_expression*).

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute len.

See also

Datatypes char, nchar, varchar, nvarchar

Functions char_length, left, str_replace

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 Java in Adaptive Server feature.

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

Examples

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

```
select license_enabled("ase_dtm")

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

Usage

- For information about installing license keys for Adaptive Server features, see your installation guide.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute license_enabled.

See also

Documents Installation guide for your platform

System procedure sp_configure

list_appcontext

Description

Lists all the attributes of all the contexts in the current session. list_appcontext is a built-in function provided by the Application Context Facility (ACF).

Syntax

list_appcontext(["*context_name*"])

Parameters

context_name
is an optional argument that names all the application context attributes in the session.

Examples

Example 1 Shows the results when a user with appropriate permissions attempts to list the application contexts:

```
select list_appcontext ([context_name])
```

```
Context Name: (CONTEXT1)
Attribute Name: (ATTR1) Value: (VALUE2)
Context Name: (CONTEXT2)
Attribute Name: (ATTR1) Value: (VALUE1)
```

Example 2 Shows the results when a user without appropriate permissions attempts to list the application contexts:

```
select list_appcontext()
```

```
Select permission denied on built-in list_appcontext,
database DBID
--------
-1
```

Usage

- This function returns 0 for success.
- Since built-in functions do not return multiple result sets, the client application receives list_appcontext returns as messages.

Standards

ANSI SQL – Compliance level: Transact-SQL extension

Permissions

Permissions depend on the user profile and the application profile, and are stored by the ACF.

See also

For more information on the ACF, see “Row-level access control” in Chapter 11, “Managing User Permissions” of the *System Administration Guide*.

Functions get_appcontext, list_appcontext, rm_appcontext, set_appcontext

lockscheme

Description

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

Syntax

lockscheme(*object_name*)

lockscheme(*object_id* [, *db_id*])

Parameters

object_name
is the name of the object that the locking scheme returns. *object_name* can also be a fully qualified name.

db_id
the ID of the database specified by *object_id*.

object_id
the ID of the object that the locking scheme returns.

Examples

Example 1 Selects the locking scheme for the titles table in the current database:

```
select lockscheme("titles")
```

Example 2 Selects the locking scheme for *object_id* 224000798 (in this case, the titles table) from database ID 4 (the pubs2 database):

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

Example 3 Returns the locking scheme for the titles table (*object_name* in this example is fully qualified):

```
select lockscheme(tempdb.ownerjoe.titles)
```

Usage

- lockscheme returns varchar(11) and allows NULLs.
- lockscheme defaults to the current database if you:
  - Do not provide a fully qualified *object_name*.
  - Do not provide a *db_id*.
  - Provide a null for *db_id*.
- If the specified object is not a table, lockscheme returns the string "not a table."

Standards

ANSI SQL – Compliance 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 74.

Standards

ANSI SQL – Compliance 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 74.

Standards ANSI SQL – Compliance 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 76.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute lower.

See also

Function 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 lowercase Unicode equivalent of the specified expression. Characters in the expression that have no lowercase equivalent are left unmodified.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute ltrim.

See also
For general information about string functions, see "String functions" on page 76.

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

```
select max(discount) from salesdetail

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

Example 2 Returns the maximum value in the discount column of the salesdetail table as a new row:

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

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 54.
- You can use max with exact and approximate numeric, character, and datetime columns; you cannot use it 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, and 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
ANSI SQL – Compliance 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 359.

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 54.
- You can use min with numeric, character, time, and datetime columns; you cannot use it with bit columns. With character columns, min finds the lowest value in the sort sequence. min implicitly converts char datatypes to varchar, and 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
ANSI SQL – Compliance 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

month

Description
Returns an integer that represents the month in the datepart of a specified date.

Syntax
month(*date_expression*)

Parameters
date_expression
is an expression of type datetime, smalldatetime, date, or a character string in a datetime format.

Examples
Returns the integer 11:

```
day("11/02/03")
----------
11
```

Usage
month(*date_expression*) is equivalent to datepart(*mm*, *date_expression*).

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute month.

See also
Datatypes datetime, smalldatetime, date

Functions datepart, day, year

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
Shows that the admin and supervisor roles are mutually exclusive:

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

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

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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute mut_excl_roles.

See also
For general information about system functions, see “System functions” on page 77.

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

newid

Description

Generates human-readable, globally unique IDs (GUIDs) in two different formats, based on arguments you provide. The length of the human-readable format of the GUID value is either 32 bytes (with no dashes) or 36 bytes (with dashes).

Syntax

newid([*optionflag*])

Parameters

option flag

- 0, or no value – the GUID generated is human-readable (varchar), but does not include dashes. This argument, which is the default, is useful for converting values into varbinary.
- -1 – the GUID generated is human-readable (varchar) and includes dashes.
- -0x0 – returns the GUID as a varbinary.
- Any other value for newid returns NULL.

Examples

Example 1 Creates a table with varchar columns 32 bytes long, then uses newid with no arguments with the insert statement:

```
create table t (UUID varchar(32))
go
insert into t values (newid())
insert into t values (newid())
go
select * from t

UUID
--------------------------------
f81d4fae7dec11d0a76500a0c91e6bf6
7cd5b7769df75cefe040800208254639
```

Example 2 Produces a GUID that includes dashes:

```
select newid(1)
------------------------------------
b59462af-a55b-469d-a79f-1d6c3c1e19e3
```

Example 3 Creates a default that converts the GUID format without dashes to a varbinary(16) column:

```
create table t (UUID_VC varchar(32), UUID
varbinary(16))
go
create default default_guid
as
```

```
strtobin(newid())
go
sp_bindefault default_guid, "t.UUID"
go
insert t (UUID_VC) values (newid())
go
```

Example 4 Returns a new GUID of type varbinary for every row that is returned from the query:

```
select newid(0x0) from sysobjects
```

Example 5 Uses newid with the varbinary datatype:

```
sp_addtype binguid, "varbinary(16)"
create default binguid_dflt
as
newid(0x0)
sp_bindefault "binguid_dflt","binguid"
create table T1 (empname char(60), empid int, emp_guid
binguid)
insert T1 (empname, empid) values ("John Doe", 1)
insert T1 (empname, empid( values ("Jane Doe", 2)
```

Usage

- newid generates two values for the globally unique ID (GUID) based on arguments you pass to newid. The default argument generates GUIDs without dashes. By default newid returns new values for every filtered row.
- You can use newid in defaults, rules, and triggers, similar to other functions.
- Make sure the length of the varchar column is at least 32 bytes for the GUID format without dashes, and at least 36 bytes for the GUID format with dashes. The column length is truncated if it is not declared with these minimum required lengths. Truncation increases the probability of duplicate values.
- An argument of zero is equivalent to the default.
- You can use the GUID format without dashes with the strtobin function to convert the GUID value to 16-byte binary data. However, using strtobin with the GUID format with dashes results in NULL values.
- Because GUIDs are globally unique, they can be transported across domains without generating duplicates.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute newid.

next_identity

Description

Retrieves the next identity value that is available for the next insert.

Syntax

next_identity(*table_name*)

Parameters

table_name
identifies the table being used.

Examples

Updates the value of c2 to 10. The next available value is 11.

```
select next_identity ("t1")
t1
--------
11
```

Usage

- next_identity returns the next value to be inserted by this task. In some cases, if multiple users are inserting values into the same table, the actual value reported as the next value to be inserted is different from the actual value inserted if another user performs an intermediate insert.
- next_identity returns a varchar chararcter to support any precision of the identity column. If the table is a proxy table, a non-user table, or the table does not have identity property, NULL is returned.

Permissions

Only the table owner, System Administrator, or database administrator can issue this command.

nullif

Description

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

Syntax

nullif(*expression*, *expression*)

Parameters

nullif
: compares the values of the two expressions. If the first expression equals the second expression, nullif returns NULL. If the first expression does not equal the second expression, nullif returns the first expression.

expression
: is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see "Expressions" on page 359.

Examples

Example 1 Selects the titles and type from the titles table. If the book type is UNDECIDED, nullif returns a NULL value:

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

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

```
select title,
    case
        when type = "UNDECIDED" then NULL
        else type
    end
from titles
```

Usage

- nullif expression alternate for a case expression.
- nullif expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a when...then construct.
- You can use nullif expressions anywhere an expression can be used in SQL.
- At least one result of the case expression must return a non-null value. For example the following results in an error message:

```
select price, coalesce (NULL, NULL, NULL)
from titles
All result expressions in a CASE expression must not be NULL.
```

- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in “Datatypes of mixed-mode expressions” on page 7. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Anyone can execute nullif.

See also — **Commands** case, coalesce, select, if...else, where clause

object_attr

Description

Reports the table's current logging mode, depending on the session, table and database-wide settings.

Syntax

object_attr(*table_name*, *string*)

Parameters

table_name
name of a table.

string
is the name of the property of the table which is been queried for. The supported string values are:

- dml_logging – returns the DML logging level for the requested object in effect based on the explicitly set table or database's DML logging level.
- dml_logging for session – returns the DML logging level for the current session, taking into account the user running object_attr, the table's schema, and rules regarding multistatement transactions, and so on. The return value from this argument can be different for different users, and different for statements or transactions for the same user.
- help – prints a list of supported string arguments.

Examples

Example 1 To determine which properties they can query, the user runs:

```
select object_attr('sysobjects', 'help')
Usage: object_attr('tabname', 'attribute')

List of options in attributes table:
     0 : help
     1 : dml_logging
     2 : dml_logging for session
```

`dml_logging` reports the statically-defined dml_logging level for the object, and `dml_logging for session` reports the run-time logging level chosen for the object depending on the database-specific and session settings.

Example 2 The default logging mode of a table with durability set to full:

```
select object_attr("pubs2..authors",
          "dml_logging")

Returns: FULL
```

Example 3 If the session has logging disabled for all tables, the logging mode returned for tables owned by this user is minimal.

```
select object_attr("pubs2..authors",
```

```
         "dml_logging")

Returns: FULL

SET DML_LOGGING MINIMAL
go

select object_attr("pubs2..authors",
         "dml_logging for session")

Returns: MINIMAL
```

Example 4 If a table has been altered to explicitly select minimal logging, object_attr returns a value of minimal, even if the session and database-wide logging is FULL.

```
create database testdb WITH DML_LOGGING = FULL
go

create table non_logged_table (...)
WITH DML_LOGGING = MINIMAL
go

select object_attr("non_logged_table",
         "dml_logging")
Returns: MINIMAL
```

Example 5 Changes a table's logging from full to minimal. If you explicitly create a table with full logging, you can reset the logging to minimal during a session if you are the table owner or a user with the sa_role:

1 Create the testdb database with minimal logging:

   ```
   create database testdb
   with dml_logging = minimal
   ```

2 Create a table with dml_logging set to full:

   ```
   create table logged_table(...)
   with dml_logging = full
   ```

3 Reset the logging for the session to minimal:

   ```
   set dml_logging minimal
   ```

4 The logging for the table is minimal:

   ```
   select object_attr("logged_table",
            "dml_logging for session")
   --------------------
   ```

```
minimal
```

Example 6 If you create a table without specifying the logging mode, changing the session's logging mode also changes the table's logging mode:

- Create the table normal_table:

  ```
  create table normal_table
  ```

- Check the session's logging:

  ```
  select object_attr("normal_table", "dml_logging")
  -------------------------
  FULL
  ```

- Set the session logging to minimal:

  ```
  set dml_logging minimal
  ```

- The table's logging is set to minimal:

  ```
  select object_attr("normal_table",
           "dml_logging for session")
  -----------------------------
  minmimal
  ```

Example 7 The logging mode returned by object_attr depends on the table you run it against. In this example, user joe runs a script, but the logging mode Adaptive Server returns changes. The tables joe.own_table and mary.other_table use a full logging mode:

```
select object_attr("own_table","dml_logging")
--------------
          FULL
```

When joe runs object_attr against mary.other_table, this table is also set to full:

```
select object_attr("mary.other_table", "dml_logging")
--------------
          FULL
```

If joe changes the dml_logging to minimal, only the logging mode of the tables he owns are affected:

```
set dml_logging minimal
select object_attr("own_table", "dml_logging for
session")
--------------
       MINIMAL
```

Tables owned by other users will continue to operate in their default logging mode:

```
Select object_attr("mary.other_table", "dml_logging for
session")
--------------
          FULL
```

Example 8 Identify the run-time choices of logging a new show_exec_info, and use it in the SQL batch:

1 Enable set showplan:

```
set showplan on
```

2 Enable the set command:

```
set show_exec_info on
```

3 Set dml_logging to minimal and check the logging with object_attr:

```
set dml_logging minimal
select object_attr("logged_table", "dml_logging for session")
```

4 Delete rows from the table:

```
delete logged_table
```

Adaptive Server reports the table's logging mode at run-time with show_exec_info parameter.

Usage

- The return type is a varchar, which appropriately returns the value of the property (for example, on or off) depending on the property queried for.
- The logging mode as reported by extensions to showplan output might be affected at run-time, if there are set statements in the same batch, preceding the execution of the DML, which changes the logging mode of the table
- The return value is the value NULL (not the string "NULL") for an unknown property.
- A special-type of string parameter, help prints to the session's output all the currently supported properties for object_attr. This allows you to quickly identify which properties are supported by object_attr.

Permissions

See also

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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute object_id.

See also
For general information about system functions, see "System functions" on page 77.

Functions col_name, db_id, object_name

System procedure sp_help

object_name

Description

Returns the name of the object with the object ID you specify; can be up to 255 bytes in length.

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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute object_name.

See also

For general information about system functions, see "System functions" on page 77.

Functions col_name, db_id, object_id

System procedure sp_help

object_owner_id

Description

Returns an object's owner ID.

Syntax

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

Parameters

object_id
is the ID of the object you are investigating.

database_id
is the ID of the database in which the object resides.

Examples

Selects the owner's ID for an object with an ID of 1, in the database with the ID of 1 (the master database):

```
select object_owner_id(1,1)
```

Permissions

Any user can execute object_owner_id.

pagesize

Description

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

Syntax

pagesize(*object_name*[,])

pagesize(*object_id*[,*db_id*[, *index_id*]])

Parameters

object_name
is the object name of the page size of this function returns.

index_name
indicates the index name of the page size you want returned.

object_id
is the object ID of the page size this function returns.

db_id
is the database ID of the object.

index_id
is the index ID of the object you want returned.

Examples

Example 1 Selects the page size for the title_id index in the current database.

```
select pagesize("title", "title_id")
```

Example 2 Returns the page size of the data layer for the object with *object_id* 1234 and the database with a *db_id* of 2 (the previous example defaults to the current database):

```
select pagesize(1234,2, null)
select pagesize(1234,2)
select pagesize(1234)
```

Example 3 All default to the current database:

```
select pagesize(1234, null, 2)
select pagesize(1234)
```

Example 4 Selects the page size for the titles table (object_id 224000798) from the pubs2 database (db_id 4):

```
select pagesize(224000798, 4)
```

Example 5 Returns the page size for the nonclustered index's pages table mytable, residing in the current database:

```
pagesize(object_id('mytable'), NULL, 2)
```

Example 6 Returns the page size for object titles_clustindex from the current database:

```
select pagesize("titles", "titles_clustindex")
```

Usage

- pagesize defaults to the data layer if you do not provide an index name or *index_id* (for example, `select pagesize("t1")`) if you use the word “null” as a parameter (for example, `select pagesize("t1", null)`.
- If the specified object is not an object requiring physical data storage for pages (for example, if you provide the name of a view), pagesize returns 0.
- If the specified object does not exist, pagesize returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute pagesize.

partition_id

Description

Returns the partition ID of the specified data or index partition name.

Syntax

partition_id(*table_name*, *partition_name*[,*index_name*])

Parameters

table_name
is the name for a table.

partition_name
is the partition name for a table partition or an index partition.

index_name
is the name of the index of interest.

Examples

Example 1 Returns the partition ID corresponding to the partition name testtable_ptn1 and index id 0 (the base table). The testtable must exist in the current database:

```
select partition_id("testtable", "testtable_ptn1")
```

Example 2 Returns the partition ID corresponding to the partition name testtable_clust_ptn1 for the index name clust_index1. The testtable must exist in the current database:

```
select partition_id("testtable", "testtable_clust_ptn1", "clust_index1")
```

Example 3 This is the same as the previous example, except that the user need not be in the same database as where the target table is located:

```
select partition_id("mydb.dbo.testtable", "testtable_clust_ptn1",
    "clust_index1")
```

Usage

You must enclose *table_name*, *partition_name* and *index_name* in quotes.

See also

Functions data_pages, object_id, partition_name, reserved_pages, row_count, used_pages

partition_name

Description

The explicit name of a new partition, partition_name returns the partition name of the specified data or index partition id.

Syntax

partition_name(*indid*, *ptnid*[, *dbid*])

Parameters

indid
is the index ID for the target partition.

ptnid
is the ID of the target partition.

dbid
is the database ID for the target partition. If you do not specify this parameter, the target partition is assumed to be in the current database.

Examples

Example 1 Returns the partition name for the given partition ID belonging to the base table (with an index ID of 0). The lookup is done in the current database because it does not specify a database ID:

```
select partition_name(0, 1111111111)
```

Example 2 Returns the partition name for the given partition ID belonging to the clustered index (index ID of 1 is specified) in the testdb database.

```
select partition_name(1, 1212121212, db_id("testdb")
```

Usage

- If the search does not find the target partition, the return is NULL.

See also

Functions data_pages, object_id, partition_id, reserved_pages, row_count

partition_object_id

Description

Displays the object ID for a specified partition ID and database ID.

Syntax

partition_object_id(*partition_id* [, *database_id*])

Parameters

partition_id
is the ID of the partition whose object ID is to be retrieved.

database_id
is the database ID of the partition.

Examples

Example 1 Displays the object ID for the partition whose partition ID is 2:

```
select partition_object_id(2)
```

Example 2 Displays the object ID for the partition whose partition ID is 14 and whose database ID is 7:

```
select partition_object_id(14,7)
```

Example 3 Returns a NULL value for the database ID because a NULL value is passed to the function:

```
select partition_object_id( 1424005073, NULL)

 -----------
NULL
(1 row affected)
```

Usage

- partition_object_id uses the current database ID if you do not include a database ID.
- partition_object_id returns NULL if you use a NULL value for the *partition_id*.
- partition_object_id returns a NULL value if you include a NULL value for database ID.
- partition_object_id returns NULL if you provide an invalid or non-existent *partition_id* or *database_id*.

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, see "Pattern matching with wildcard characters" on page 377.

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 Selects the author ID and the starting character position of the word "circus" in the copy column:

```
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
```

Example 2

```
select au_id, patindex("%circus%", copy,
```

```
     using chars)
from blurbs
```

Example 3 Finds all the rows in sysobjects that start with "sys" with a fourth character that is "a", "b", "c", or "d":

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

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

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 0 if *pattern* is not found.
- You can use patindex on all character data, including text and image data.
- For unichar, univarchar, and unitext, patindex returns the offset in Unicode characters. The pattern string is implicitly converted to UTF-16 before comparison, and the comparison is based on the default unicode sort order configuration. For example, this is what is returned if a unitext column contains row value U+0041U+0042U+d800U+dc00U+0043:

  ```
  select patindex("%C%", ut) from unitable
  -----------
  4
  ```

- 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, patindex returns 0.

- If you give a varchar expression 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 76.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute patindex.

See also — **Functions** charindex, substring

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.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute pi.

See also — For general information about mathematical functions, see “Mathematical functions” on page 74.

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*.

  In expressions of type numeric or decimal, this function returns precision:38, scale 18.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute power.

See also

For general information about mathematical functions, see “Mathematical functions” on page 74.

Functions exp, log, log10

proc_role

Description

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

Note Sybase supports—and recommends—that you use has_role instead of proc_role. You need not, however, convert your existing uses of proc_role to has_role.

Syntax

proc_role("*role_name*")

Parameters

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

Examples

Example 1 Creates a procedure to check if the user is a System Administrator:

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

Example 2 Checks that the user has been granted the System Security Officer role:

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

Example 3 Checks that the user has been granted the Operator role:

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

Usage

- Using proc_role with a procedure that starts with “sp_” returns an error.
- proc_role, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- proc_role returns 0 if the user has:
  - Not been granted the specified role
  - Not been granted a role which contains the specified role
  - 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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute proc_role.

See also

For general information about system functions, see “System functions” on page 77.

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

Functions mut_excl_roles, role_contain, role_id, role_name, show_role

pssinfo

Description — Returns information from the Adaptive Server process status structure (pss).

Syntax — pssinfo(*spid* | 0, '*pss_field*')

Parameters

spid
: is the process ID. When you enter 0, the current process is used.

pss_field
: is the process status structure field. Valid values are:

- dn – distinguished name when using LDAP authentication.
- extusername – when using external authentication like (PAM, LDAP), extusername returns the external PAM or LDAP user name used.
- ipaddr – client IP address.
- ipport – client IP port number used for the client connection associated with the user task being queried.
- isolation_level – isolation level for the current session.
- tempdb_pages – number of tempdb pages used.

Examples — Displays the port number for spid number 14

```
select pssinfo(14,'ipport')
----------------
           52039
```

Usage

- The pssinfo function also includes the option to display the external user name and the distinguish name.
- ipport output, combined with ipaddr output, allows you to uniquely identify network traffic between Adaptive Server and the client.

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*.

  To express numeric or decimal dataypes, this function returns precision: 38, scale 18.

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

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute radians.

See also

For general information about mathematical functions, see “Mathematical functions” on page 74.

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 pseudorandom 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 pseudorandom 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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute rand.

See also

For general information about mathematical functions, see “Mathematical functions” on page 74.

Datatypes Approximate numeric datatypes

Functions rand2

rand2

Description

Returns a random value between 0 and 1, which is generated using the specified seed value, and computed for each returned row when used in the select list.

Syntax

rand2([*integer*])

Parameters

integer
is any integer (tinyint, smallint, or int) column name, variable, constant expression, or a combination of these.

Examples

If there are n rows is table t, the following select statement returns n different random values, not just one.

```
select rand2() from t
------------
```

Usage

- rand2, a mathematical function, returns a random float value between 0 and 1, using the optional integer as a seed value. Unlike rand, it is computed for each returned row when it is used in the select list.
- The behavior of rand2 in places other than the select list is currently undefined.
- For more information about the 32-bit pseudorandom integer generator, see the Usage section of rand.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute rand.

See also

For general information about mathematical functions, see "Mathematical functions" on page 74.

Datatypes Approximate numeric datatypes

Functions rand

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 fits into 16K, whichever is less.
- If *char_expr* or *uchar_expr* is NULL, returns a single NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute replicate.

See also

For general information about string functions, see "String functions" on page 76.

Function stuff

reserve_identity

Description

reserve_identity allows a process to reserve a block of identity values for use by that process.

After a process calls reserve_identity to reserve the block of values, subsequent identity values needed by this process are drawn from this reserved pool. When these reserved numbers are exhausted, or if you insert data into a different table, the existing identity options apply. reserve_identity can retain more than one block of identity values, so if inserts to different tables are interleaved by a single process, the next value in a table's reserved block is used.

Reserves a specified size block of identity values for the specified table, which are used exclusively by the calling process. Returns the reserved starting number, and subsequent inserts into the specified table by this process use these values. When the process terminates, any unused values are eliminated.

Syntax

reserve_identity (*table_name*, *number_of_values*)

Parameters

table_name
is the name of the table for which the reservation are made. The name can be fully qualified; that is, it can include the *database_name*, *owner_name*, and *object_name* (in quotes).

number_of_values
is the number of sequential identity values reserved for this process. This must be a positive value that will not cause any of the reserved values to exceed the maximum values for the datatype of the identity column.

Examples

Describes a typical usage scenario for reserve_identity, and assumes that table1 includes col1 (with a datatype of int) and a col2 (an identity column with a datatype of int). This process is for spid 3:

```
select reserve_identity( table1, 5 )
---------------
10
```

Insert values for spids 3 and 4:

```
Insert table1 values(56) -> spid 3
Insert table1 values(48) -> spid 3
Insert table1 values(96) -> spid 3
Insert table1 values(02) -> spid 4
Insert table1 values(84) -> spid 3
```

Select from table table1:

```
select * from table1
```

```
Col1          col2
--------       -----
3              1-> spid 3 reserved 1-5
3              2-> spid 3
3              3-> spid 3
4              6<= spid 4 gets next unreserved value
3              4<= spid 3 continues with reservation
```

The result set shows that spid 3 reservered identity values 1 – 5, spid 4 receives the next unreserved value, and then spid 3 reserves the subsequent identity values.

Usage

- The return value, *start_value*, is the starting value for the block of reserved identity values. The calling process uses this value for the next insert into the specified table
- reserve_identity allows a process to:
  - Reserve identity values without issuing an insert statement.
  - Know the values reserved prior issuing the insert statement
  - “Grab” different size blocks of identity values, according to need.
  - Better control “over gaps” by reserving only what is needed (that is, they are not restricted by preset server grab size
- Values are automatically used with no change to the insert syntax.
- NULL values are returned if:
  - A negative value or zero is specified as the block size.
  - The table does not exist.
  - The table does not contain an identity column.
- If you issue reserve_identity on a table in which this process has already reserved these identity values, the function succeeds and the most recent group of values is used.
- You cannot use reserve_identity to reserve identity values on a proxy table. Local servers can use reserve_identity on a remote table if the local server calls a remote procedure that calls reserve_identity. Because these reserved values are stored on the remote server but in the session belonging to the local server, subsequent inserts to the remote table use the reserved values.

- If the identity_gap is less than the reserved block size, the reservation succeeds by reserving the specified block size (not an identity_gap size) of values. If these values are not used by the process, this results in potential gaps of up to the specified block size regardless of the identity_gap setting.

Permissions

You must have insert permission to reserve identity values.

reserved_pages

Description

Reports the number of pages reserved for a database, object, or index. The result includes pages used for internal structures.

This function replaces the old reserved_pgs function used in Adaptive Server versions earlier than 15.0.

Syntax

reserved_pages(*dbid*, *object_id*[, *indid*[, *ptnid*]])

Parameters

dbid
is the database ID of the database where the target object resides.

object_id
is an object ID for a table.

indid
is the index ID of target index.

ptnid
is the partition ID of target partition.

Examples

Example 1 Returns the number of pages reserved by the object with a object ID of 31000114 in the specified database (including any indexes):

```
select reserved_pages(5, 31000114)
```

Example 2 Returns the number of pages reserved by the object in the data layer, regardless of whether or not a clustered index exists:

```
select reserved_pages(5, 31000114, 0)
```

Example 3 Returns the number of pages reserved by the object in the index layer for a clustered index. This does not include the pages used by the data layer:

```
select reserved_pages(5, 31000114, 1)
```

Example 4 Returns the number of pages reserved by the object in the data layer of the specific partition, which in this case is 2323242432:

```
select reserved_pages(5, 31000114, 0, 2323242432)
```

Example 5 Use one of the following three methods to calculate space in a database with reserved_pages:

- Use case expressions to select a value appropriate for the index you are inspecting, selecting all non-log indexes in sysindexes for this database. In this query:

- The data has a value of “index 0”, and is available when you include the statements `when sysindexes.indid = 0` or `sysindexes.indid = 1`.
- indid values greater than 1 for are indexes. Because this query does not sum the data space into the index count, it does not include a page count for indid of 0.
- Each object has an index entry for index of 0 or 1, never both.
- This query counts index 0 exactly once per table.

```
select
'data rsvd' = sum( case
          when indid > 1 then 0
          else reserved_pages(db_id(), id, 0)
          end ),
'index rsvd' = sum( case
          when indid = 0 then 0
          else reserved_pages(db_id(), id, indid)
          end )
from sysindexes
where id != 8
data rsvd   index rsvd
----------   -----------
       812           1044
```

- Query sysindexes multiple times to display results after all queries are complete:

```
declare @data int,
@dbsize int,
@dataused int,
@indices int,
@indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
     @dataused = sum( used_pages(db_id(), id, 0) )
from sysindexes
where id != 8
and indid <= 1
select @indices = sum( reserved_pages(db_id(), id, indid) ),
     @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8 and indid > 0
select @dbsize   as 'db size',
@data as 'data rsvd'
db size    data rsvd
----------- -----------
       NULL         820
```

- Query sysobjects for data space information and sysindexes for index information. From sysobjects, select table objects: [S]ystem or [U]ser:

```
declare   @data int,
          @dbsize int,
          @dataused int,
          @indices int,
          @indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
@dataused = sum( used_pages(db_id(), id, 0) )
from sysobjects
where id != 8
and type in ('S', 'U')
select @indices = sum( reserved_pages(db_id(), id, indid) ),
       @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8
and indid > 0
select   @dbsize as 'db size',
         @data as 'data rsvd',
         @dataused as 'data used',
         @indices as 'index rsvd',
         @indused as 'index used'
db size    data rsvd    data used    index rsvd    index used
---------  -----------  -----------  -----------   ----------
     NULL          812          499         1044          381
```

Usage

- If a clustered index exists on an all-pages locked table, passing an index ID of 0 reports the reserved data pages, and passing an index ID of 1 reports the reserved index pages. All erroneous conditions result in a value of zero being returned.
- reserved_pages counts whatever you specify; if you supply a valid database, object, index (data is "index 0" for every table), it returns the reserved space for this database, object, or index. However, it can also count a database, object, or index multiple times. If you have it count the data space for every index in a table with multiple indexes, you get it counts the data space once for every index. If you sum these results, you get the number of indexes multiplied by the total data space, not the total number of data pages in the object.
- For Adaptive Server version 15.0, reserved_pages replaces the reserved_pgs function. These are the differences between reserved_pages and reserved_pgs.

- In Adaptive Server versions 12.5 and earlier, Adaptive Server stored OAM pages for the data and index in sysindexes. In Adaptive Server versions 15.0 and later, this information is stored per-partition in sysparitions. Because this information is stored differently, reserved_pages and reserved_pgs require different parameters and have different result sets.
- reserved_pgs required a page ID. If you supplied a value that did not have a matching sysindexes row, the supplied page ID was 0 (for example, the data OAM page of a nonclustered index row). Because 0 was never a valid OAM page, if you supplied a page ID of 0, reserved_pgs returned 0; because the input value is invalid, reserved_pgs could not count anything.

  However, reserved_pages requires an index ID, and 0 is a valid index ID (for example, data is "index 0" for every table). Because reserved_pages can not tell from the context that you do not require it to recount the data space for any index row except indid 0 or 1, it counts the data space every time you pass 0 as an index ID. Because reserved_pages counts this data space once per row, its yields a sum many times the true value.

  These differences are described as:

  - reserved_pgs does not affect the sum if you supply 0 as a value for the page ID for the OAM page input; it just returns a value of 0.
  - If you supply reserved_pages with a value of 0 as the index ID, it counts the data space. Issue reserved_pages only when you want to count the data or you will affect the sum.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute reserved_pgs.

See also

Command update statistics

Function data_pages, reserved_pages, row_count, used_pages

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, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute reverse.

See also
For general information about string functions, see “String functions” on page 76.

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, right returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension

Permissions

Any user can execute right.

See also

For general information about string functions, see “String functions” on page 76.

Functions rtrim, substring

rm_appcontext

Description

Removes a specific application context, or all application contexts. rm_appcontext is a function provided by the Application Context Facility (ACF).

Syntax

rm_appcontext("*context_name*", "*attribute_name*")

Parameters

context_name
is a row specifying an application context name. It is saved as datatype char(30).

attribute_name
is a row specifying an application context attribute name. It is saved as datatype char(30).

Examples

Example 1 Removes an application context by specifying some or all attributes:

```
select rm_appcontext("CONTEXT1", "*")
----------
0

select rm_appcontext("*", "*")
---------
0

select rm_appcontext("NON_EXISTING_CTX","ATTR")
---------
-1
```

Example 2 Shows the result when a user without appropriate permissions attempts to remove an application context:

```
select rm_appcontext("CONTEXT1","ATTR2")
----------
-1
```

Usage

- This function always returns 0 for success.
- All the arguments for this function are required.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Permissions depend on the user profile and the application profile, which are stored by ACF.

See also

For more information on the ACF see "Row-level access control" in Chapter 11, "Managing User Permissions" of the *System Administration Guide*.

Functions get_appcontext, list_appcontext, set_appcontext

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*.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute role_contain.

See also

For more information about system functions, see "System functions" on page 77.

Documents For more information about contained roles and role hierarchies, see the *System Administration Guide*.

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 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 sysssrvroles system table.

Examples

Example 1 Returns the system role ID of sa_role:

```
select role_id("sa_role")
------
0
```

Example 2 Returns the system role ID of the “intern_role”:

```
select role_id("intern_role")
------
6
```

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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute role_id.

See also

For more information about system functions, see “System functions” on page 77.

Documents For more information about roles, see the *System Administration Guide*.

Functions mut_excl_roles,proc_role,role_contain, role_name

role_name

Description — Returns the name of a 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.

Standards — ANSI SQL – Compliance level: Transact-SQL extension

Permissions — Any user can execute role_name.

See also — For more information about system functions, see "System functions" on page 77.

Functions mut_excl_roles, proc_role, role_contain, role_id

round

Description

Returns the value of the specified number, rounded to a specified number of decimal places.

Syntax

round(*number*, *decimal_places*)

Parameters

number
is any exact numeric (numeric, dec, decimal, tinyint, smallint, int, or bigint), 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 value for *decimal_places* determines the number of significant digits to the right of the decimal point; a negative value for *decimal_places* determines 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 specified for *number*, Adaptive Server returns 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, the following returns a value of 0.00:

```
select round(55.55, -3)
```

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute round.

See also

For general information about mathematical functions, see “Mathematical functions” on page 74.

Functions abs, ceiling, floor, sign, str

row_count

Description

Returns an estimate of the number of rows in the specified table.

Syntax

row_count(*dbid*, *object_id* [,*ptnid*])

Parameters

dbid
is the the database ID where target object resides

object_id
is the object ID of table

ptnid
is the partition ID of interest

Examples

Example 1 Returns an estimate of the number of rows in the given object:

```
select row_count(5, 31000114)
```

Example 2 Returns an estimate of the number of rows in the specified partition (with partition ID of 2323242432) of the object with object IDof 31000114:

```
select row_count(5, 31000114, 2323242432)
```

Usage

All erroneous conditions will return in a value of zero being returned.

Standards

ANSI SQL – Compliance level: Transact-SQL extension

Permissions

Any user can execute row_count.

See also

Functions reserved_pages, used_pages

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.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute rtrim.

See also — For general information about string functions, see "String functions" on page 76.

Function ltrim

sdc_intempdbconfig

Description

Cluster environments only – returns 1 if the system is currently in temporary database configuration mode; if not, returns 0.

Syntax

sdc_intempdbconfig()

Examples

```
select sdc_intempdbconfig()
```

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can run sdc_intempdbconfig.

set_appcontext

Description

Sets an application context name, attribute name, and attribute value for a user session, defined by the attributes of a specified application. set_appcontext is a built-in function that the Application Context Facility (ACF) provides.

Syntax

set_appcontext("*context_name*, "*attribute_name*", "*attribute_value*")

Parameters

context_name
is a row that specifies an application context name. It is saved as the datatype char(30).

attribute_name
is a row that specifies an application context attribute name. It is saved as the datatype char(30).

attribute_value
is a row that specifies and application attribute value. It is saved as the datatype char(30).

Examples

Example 1 Creates an application context called CONTEXT1, with an attribute ATTR1 that has the value VALUE1.

```
select set_appcontext ("CONTEXT1", "ATTR1", "VALUE1")

---------------
0
```

Attempting to override the existing application context created causes:

```
select set_appcontext("CONTEXT1", "ATTR1", "VALUE1")

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

Example 2 Shows set_appcontext including a datatype conversion in the value.

```
declare@numericvarchar varchar(25)
select @numericvar = "20"
select set_appcontext ("CONTEXT1", "ATTR2",
convert(char(20), @numericvar))

------------
0
```

Example 3 Shows the result when a user without appropriate permissions attempts to set the application context.

```
select set_appcontext("CONTEXT1", "ATTR2", "VALUE1")
--------------
-1
```

Usage

- set_appcontext returns 0 for success and -1 for failure.
- If you set values that already exist in the current session, set_appcontext returns -1.
- This function cannot override the values of an existing application context. To assign new values to a context, remove the context and re-create it using new values.
- set_appcontext saves attributes as char datatypes. If you are creating an access rule that must compare the attribute value to another datatype, the rule should convert the char data to the appropriate datatype.
- All the arguments for this function are required.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Permissions depend on the user profile and the application profile, stored by ACF.

See also

Documents For more information on the ACF see “Row-level access control” in Chapter 11, “Managing User Permissions” of the *System Administration Guide*.

Functions get_appcontext, list_appcontext, rm_appcontext

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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute show_role.

See also
For general information about system functions, see "System functions" on page 77.

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

Shows that the user's current session is encrypting data and performing replay detection checks:

```
select show_sec_services()
```

```
encryption, replay_detection
```

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

ANSI SQL – Compliance 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, int, or bigint), 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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute sign.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

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).

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute sin.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

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 2-11 on page 275 shows the valid values.

collation_ID
is an integer constant or a variable that specifies the collation to use. Table 2-11 on page 275 shows the valid values.

Examples

Example 1 Shows sorting by European language dicitionary order:

```
select * from cust_table where cust_name like "TI%" order by
    (sortkey(cust_name, "dict")
```

Example 2 Shows sorting by simplified Chinese phonetic order:

```
select *from cust_table where cust name like "TI%" order by
    (sortkey(cust-name, "gbpinyin")
```

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 Shows sorting by Simplified Chinese phonetic order using preexisting keys:

```
select * from cust_table where cust_name like "TI%" order by
    cust_chinese_sort.
```

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. Ro retrieve the character data in the desired order, include in the select statement 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.

- sortkey can generate up to sixbytes of collation information for each input character. Therefore, the result from using sortkey may exceed the length limit of the varbinary datatype. If this happens, the result is truncated to fit. Since this limit is dependent on the logical page size of your server, truncation removes result bytes for each input character until the result string is less than the following for DOL and APL tables:

Table 2-10: Maximum row and column length—APL and DOL tables

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

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

If this occurs, Adaptive Server issues a warning message, but the query or transaction that contained the sortkey function continues to run.

- *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, Adaptive Server uses the binary sort order.
- If you do not specify a value for *collation_name* or *collation_ID*, sortkey assumes binary collation.
- The binary values generated from the sortkey function can change from one major version to another major version of Adaptive Server, such as version 12.0 to 12.5, version 12.9.2 to 12.0, and so on. If you are upgrading to the current version of Adaptive Server, regenerate keys and repopulate the shadow columns before any binary comparison takes place.

  Note Upgrades from version 12.5 to 12.5.0.1 do not require this step, and Adaptive Server does not generate any errors or warning messages if you do not regenerate the keys. Although a query involving the shadow columns should work fine, the comparison result may differ from the pre-upgrade server.

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 versions of Adaptive Server later than 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 files are located in *$SYBASE/collate/unicode*.

Both of these methods work equally well, but a "built-in" table is tied to a Adaptive Server database, while an external table is not. If you use an Adaptive Server database, a built-in table provides the best performance. Both methods can handle any mix of English, European, and Asian languages.

There are two ways to use 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. However, this method generates sort keys on-the-fly, and therefore does not provide optimum performance on large data sets of moe than 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 such as French, Chinese, and so on. 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 the list by executing either sp_helpsort, or by querying and selecting the name, id, and description from syscharsets (type is between 2003 and 2999).

- Table 2-11 lists the valid values for *collation_name* and *collation_ID*.

Table 2-11: Collation names and IDs

Description	Collation name	Collation ID
Deafult Unicode multilingual	default	20
Thai dictionary order	thaidict	21
ISO14651 standard	iso14651	22
UTF-16 ordering – matches UTF-8 binary ordering	utf8bin	24
CP 850 Alternative – no accent	altnoacc	39
CP 850 Alternative – lowercase first	altdict	45
CP 850 Western European – no case preference	altnocsp	46
CP 850 Scandinavian – dictionary ordering	scandict	47
CP 850 Scandinavian – case-insensitive with preference	scannocp	48
GB Pinyin	gbpinyin	n/a
Binary sort	binary	50
Latin-1 English, French, German dictionary	dict	51
Latin-1 English, French, German no case	nocase	52

Description	Collation name	Collation ID
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 Russian dictionary	rusdict	58
ISO 8859-5 Russian no case	rusnocs	59
ISO 8859-5 Cyrillic dictionary	cyrdict	63
ISO 8859-5 Cyrillic no case	cyrnocs	64
ISO 8859-7 Greek dictionary	elldict	65
ISO 8859-2 Hungarian dictionary	hundict	69
ISO 8859-2 Hungarian no accents	hunnoac	70
ISO 8859-2 Hungarian no case	hunnocs	71
ISO 8859-9 Turkish dictionary	turdict	72
ISO 8859-9 Turkish no accents	turknoac	73
ISO 8859-9 Turkish no case	turknocs	74
CP932 binary ordering	cp932bin	129
Chinese phonetic ordering	dynix	130
GB2312 binary ordering	gb2312bn	137
Common Cyrillic dictionary	cyrdict	140
Turkish dictionary	turdict	155
EUCKSC binary ordering	eucksc bn	161
Chinese phonetic ordering	gbpinyin	163
Russian dictionary ordering	rusdict	165
SJIS binary ordering	sjisbin	179
EUCJIS binary ordering	eucjisbn	192
BIG5 binary ordering	big5bin	194
Shift-JIS binary order	sjisbin	259

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute sortkey.

See also

Function compare

soundex

Description

Returns a four-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 four-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 alphabetic 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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute soundex.

See also

For general information about string functions, see “String functions” on page 76.

Function 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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute space.

See also
For general information about string functions, see “String functions” on page 76.

Functions isnull, rtrim

spid_instance_id

Description

Cluster environments only – returns the instance ID on which the specified process id (spid) is running.

Syntax

spid_instance_id(*spid_value*)

Parameters

spid_value
the spid number whose instance id is requested

Examples

Returns the ID of the instance that is running process id number 27:

```
select spid_instance_id(27)
```

Usage

- If you do not include a spid value, spid_instance_id returns NULL.
- If you enter an invalid or non-existing process id value, spid_instance_id returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute spid_instance_id.

square

Description — Returns the square of a specified value expressed as a float.

Syntax — square(*numeric_expression*)

Parameters

numeric_expression
: is a numeric expression of type float.

Examples

Example 1 Returns the square from an integer column:

```
select square(total_sales)from titles
------------
16769025.00000
15023376.00000
350513284.00000
...
16769025.00000
(18 row(s) affected)
```

Example 2 Returns the square from a money column:

```
select square(price) from titles
-----------
399.600100
142.802500
8.940100
NULL
...
224.700100
(18 row(s) affected)
```

Usage — This function is the equivalent of power(*numeric_expression*,2), but it returns type float rather than int.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute square.

See also

Function power

Datatypes exact_numeric, approximate_numeric, money, float

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.
```

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute sqrt.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

Function power

stddev

Description

Computes the standard deviation of a sample consisting of a numeric expression, as a double.

Note stddev and stdev are aliases for stddev_samp. See stddev_samp on page 287 for details.

---

stdev

Description

Computes the standard deviation of a sample consisting of a numeric expression, as a double.

Note stddev and stdev are aliases for stddev_samp. See stddev_samp on page 287 for details.

---

stdevp

Description

Computes the standard deviation of a population consisting of a numeric expression, as a double.

Note stdevp is an alias for stddev_pop. See stddev_pop on page 285 for details.

---

stddev_pop

Description

Computes the standard deviation of a population consisting of a numeric expression, as a double. stdevp is an alias for stddev_pop, and uses the same syntax.

Syntax

stddev_pop ([all | distinct] *expression*)

Parameters

all
applies stddev_pop to all values. all is the default.

distinct
eliminates duplicate values before stddev_pop is applied.

expression
is the expression—commonly a column name—in which its population-based standard deviation is calculated over a set of rows.

Examples

The following statement lists the average and standard deviation of the advances for each type of book in the pubs2 database.

```
select type, avg(advance) as "avg", stddev_pop(advance)
  as "stddev" from titles group by type order by type
```

Usage

Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

Figure 2-3: The formula for population-related statistical aggregate functions

The formula that defines the variance of the population of size n having mean μ (var_pop) is as follows. The population standard deviation (stddev_pop) is the positive square root of this.

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{n}$$

σ^2 = Variance

n = Population size

μ = Mean of the values x_i

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute stddev_pop.

See also

For general information about aggregate functions, see “Aggregate functions” on page 54.

Functions stddev_samp, var_pop, var_samp

stddev_samp

Description

Computes the standard deviation of a sample consisting of a numeric expression, as a double. stdev and stddev are aliases for stddev_samp, and use the same syntax.

Syntax

stddev_samp ([all | distinct] *expression*)

Parameters

all
: applies stddev_samp to all values. all is the default.

distinct
: eliminates duplicate values before stddev_samp is applied.

expression
: is any numeric datatype (float, real, or double precision) expression.

Examples

The following statement lists the average and standard deviation of the advances for each type of book in the pubs2 database.

```
select type, avg(advance) as "avg",
  stddev_samp(advance) as "stddev" from titles
  where total_sales > 2000 group by type order by type
```

Usage

Computes the sample standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

Figure 2-4: The formula for sample-related statistical aggregate functions

The formula that defines an unbiased estimate of the population variance from a sample of size n having mean $\bar{x}$ (var_samp) is as follows. The sample standard deviation (stddev_samp) is the positive square root of this.

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

s^2 = Variance

n = Sample size

$\bar{x}$ = Mean of the values x_i

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute stddev_samp.

See also

For general information about aggregate functions, see “Aggregate functions” on page 54.

Functions stddev_pop, var_pop, var_samp

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 76.
- *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

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute str.

See also

Functions abs, ceiling, floor, round, sign

str_replace

Description

Replaces any instances of the second string expression (*string_expression2*) that occur within the first string expression (*string_expression1*) with a third expression (*string_expression3*).

Syntax

str_replace("*string_expression1*", "*string_expression2*", "*string_expression3*")

Parameters

string_expression1
is the source string, or the string expression to be searched, expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

string_expression2
is the pattern string, or the string expression to find within the first expression (*string_expression1*). *string_expression2* is expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

string_expression3
is the replacement string expression, expressed as char, varchar, unichar, univarchar, binary, or varbinary datatype.

Examples

Example 1 Replaces the string *def* within the string *cdefghi* with *yyy*.

```
str_replace("cdefghi","def","yyy")
-------------
cyyyghi
(1 row(s) affected)
```

Example 2 Replaces all spaces with "toyota".

```
select str_replace("chevy, ford, mercedes",
"","toyota")
----------
chevy,toyotaford,toyotamercedes
(1 row(s) affected)
```

Note Adaptive Server converts an empty string constant to a string of one space automatically, to distinguish the string from NULL values.

Example 3 Returns “abcghijklm”:

```
select str_replace("abcdefghijklm", "def", NULL)

----------
abcghijklm
(1 row affected)
```

Usage

- Returns varchar data if *string_expression* (1, 2, or 3) is char or varchar.

- Returns univarchar data if *string_expression* (1, 2, or 3) is unichar or univarchar.
- Returns varbinary data if *string_expression* (1, 2, or 3) is binary or varbinary.
- All arguments must share the same datatype.
- If any of the three arguments is NULL, the function returns null.

  str_replace accepts NULL in the third parameter and treats it as an attempt to replace *string_expression2* with NULL, effectively turning str_replace into a "string cut" operation.

  For example, the following returns "abcghijklm":

  ```
  str_replace("abcdefghijklm", "def", NULL)
  ```

- The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all arguments are variables with known constant values, Adaptive Server calculates the result length as:

  ```
  result_length = ((s/p)*(r-p)+s)
  where
  s = length of source string
  p = length of pattern string
  r = length of replacement string
  if (r-p) <= 0, result length = s
  ```

- If the source string (*string_expression1*) is a column, and *string_expression2* and *string_expression3* are constant values known at compile time, Adaptive Server calculates the result length using the formula above.
- If Adaptive Server cannot calculate the result length because the argument values are unknown when the expression is compiled, the result length used is 255, unless traceflag 244 is on. In that case, the result length is 16384.
- result_len never exceeds 16384.

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute str_replace.

See also — **Datatypes** char, varchar, binary, varbinary, unichar, univarchar

Function length

strtobin

Description

Converts a sequence of alphanumeric characters to their equivalent hexadecimal digits.

Syntax

select strtobin(“*string of valid alphanumeric characters*”)

Parameters

string of valid alphanumeric characters
is string of valid alphanumeric characters, which consists of [1 – 9], [a – f] and [A – F].

Examples

Example 1 Converts the alphanumeric string of “723ad82fe” to a sequence of hexadecimal digits:

```
select strtobin("723ad82fe")
go

-----------------------------------
0x0723ad82fe
```

The in-memory representation of the alphanumeric character string and its equivalent hexadecimal digits are:

Alphanumeric character string (9 bytes)

0	7	2	3	a	d	8	2	f	e

Hexadecimal digits (5 bytes)

0	7	2	3	a	d	8	2	f	e

The function processes characters from right to left. In this example, the number of characters in the input is odd. For this reason, the hexadecimal sequence has a prefix of “0” and is reflected in the output.

Example 2 Converts the alphanumeric string of a local variable called *@str_data* to a sequence of hexadecimal digits equivalent to the value of “723ad82fe”:

```
declare @str_data varchar(30)
select @str_data = "723ad82fe"
select strtobin(@str_data)
go

----------
0x0723ad82fe
```

Usage

- Any invalid characters in the input results in NULL as the output.
- The input sequence of hexadecimal digits must have a prefix of “0x”.
- A NULL input results in NULL output.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute strtobin.

See also **Function** bintostr

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 76.
- If the start position or the length is negative, a NULL string is returned. If the start position is zero or 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 Eexamples 2 and 3).
- If *char_expr1* or *uchar_expr1* is NULL, stuff returns NULL. If *char_expr1* or *or uchar_expr1* is a string value and *char_expr2* or *uchar_expr2* is NULL, stuff replaces the deleted characters with nothing.
- If you give a varchar expression as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

Standards

ANSI SQL – Compliance 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 Displays the last name and first initial of each author, for example, "Bennet A.":

```
select au_lname, substring(au_fname, 1, 1)
from authors
```

Example 2 Converts the author's last name to uppercase, then displays the first three characters:

```
select substring(upper(au_lname), 1, 3)
from authors
```

Example 3 Concatenates pub_id and title_id, then displays the first six characters of the resulting string:

```
select substring((pub_id + title_id), 1, 6)
from titles
```

Example 4 Extracts the lower four digits from a binary field, where each position represents two binary digits:

```
select substring(xactid,5,2)
from syslogs
```

Usage

- substring, a string function, returns part of a character or binary string. For general information about string functions, see "String functions" on page 76.
- If substring's second argument is NULL, the result is NULL. If substring's first or third argument is NULL, the result is blank..

- 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 — ANSI SQL – Compliance 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 359.

Examples

Example 1 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:

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

Example 2 Used with a group by clause, the aggregate functions produce single values for each group, rather than for the entire table. This statement produces summary values for each type of book:

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

Example 3 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:

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

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 54.

- 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.When you sum bigint data, Adaptive Server treats the result as a bigint.To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums appropriately.
- You cannot use sum with the binary datatypes.
- This function defines only numeric types; use with Unicode expressions generates an error.

Standards

ANSI SQL – Compliance 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 77.
- 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

ANSI SQL – Compliance 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_id
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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute suser_name.

See also

For general information about system functions, see "System functions" on page 77.

Functions suser_id, user_name

syb_quit

Description

Terminates the connection.

Syntax

syb_quit()

Examples

Terminates the connection in which the function is executed and returns an error message.

```
select syb_quit()

-------------
CT-LIBRARY error:
    ct_results(): network packet layer:
internal net library error: Net-Library operation
terminated due to disconnect
```

Usage

You can use syb_quit to terminate a script if the isql preprocessor command exit causes an error.

Permissions

Any user can execute syb_quit.

syb_sendmsg

Description

UNIX only 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 Sends the message "Hello" to port 3456 at IP address 120.10.20.5:

```
select syb_sendmsg("120.10.20.5", 3456, "Hello")
```

Example 2 Reads the IP address and port number from a user table, and uses a variable for the message to be sent:

```
declare @msg varchar(255)
    select @msg = "Message to send"
    select syb_sendmsg (ip_address, portnum, @msg)
    from sendports
    where username = user_name()
```

Usage

- 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 that you do not use syb_sendmsg to send sensitive information across the network. By enabling this functionality, the user accepts any security problems that result from its use.
- For a sample C program that creates a UDP port, see sp_sendmsg.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute syb_sendmsg.

See also

System procedure sp_sendmsg

sys_tempdbid

Description

Cluster environments only – returns the id of the effective local system temporary database of the specified instance. Returns the id of the effective local system temporary database of the current instance when *instance_id* is not specified.

Syntax

sys_tempdbid(*instance_id*)

Parameters

instance_id
ID of the instance.

Examples

Returns the effective local system temporary database id for the instance with an instance id of 3:

```
select sys_tempdbid(3)
```

Usage

If you do not specify an instance ID, sys_tempdbid returns the id of the effective local system temporary database for the current instance.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can run sys_tempdbid.

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).

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute tan.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

Functions atan, atn2, degrees, radians

tempdb_id

Description

Reports the temporary database to which a given session is assigned. The input of the tempdb_id function is a server process ID, and its output is the temporary database to which the process is assigned. If you do not provide a server process, tempdb_id reports the dbid of the temporary database assigned to the current process.

Syntax

tempdb_id()

Examples

Finds all the server processes that are assigned to a given temporary database:

```
select spid from master..sysprocesses
    where tempdb_id(spid) = db_id("tempdatabase")
```

Usage

select tempdb_id gives the same result as select @@*tempdbid*.

See also

Commands select

textptr

Description

Returns a pointer to the first page of a text, image, or unitext column.

Syntax

textptr(*column_name*)

Parameters

column_name
is the name of a text column.

Examples

Example 1 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 placed in local variable *@val* and supplied as a parameter to the readtext command, which returns 5 bytes, starting at the second byte (offset of 1):

```
declare @val binary(16)
    select @val = textptr(copy) from blurbs
    where au_id = "486-29-1786"
    readtext blurbs.copy @val 1 5
```

Example 2 Selects the title_id column and the 16-byte text pointer of the copy column from the blurbs table:

```
select au_id, textptr(copy) from blurbs
```

Usage

- textptr, a text and image function, returns the text pointer value, a 16-byte varbinary value.
- If a text, unitext, or 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.

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

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute textptr.

See also

For general information about text and image functions, see "Text and image functions" on page 78.

Datatypes text, image, and unitext datatypes

Function textvalid

Commands insert, update, readtext, writetext

textvalid

Description
Returns 1 if the pointer to the specified text or unitext 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
Reports whether a valid text pointer exists for each value in the blurb column of the texttest table:

```
select textvalid ("texttest.blurb", textptr(blurb))
from texttest
```

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 78.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute textvalid.

See also
Datatypes text, image, and unitext datatypes

Function 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, to_unichar returns NULL.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute to_unichar.

See also
For general information about string functions, see “String functions” on page 76.

Datatypes text, image, and unitext datatypes

Function char

tran_dumpable_status

Description

Returns a true/false indication of whether dump transaction is allowed.

Syntax

tran_dumpable_status("*database_name*")

Parameters

database_name
is the name of the target database.

Examples

Checks to see if the pubs2 database can be dumped:

```
1> select tran_dumpable_status("pubs2")
2> go

 -----------
         106

(1 row affected)
```

In this example, you cannot dump pubs2. The return code of 106 is a sum of all the conditions met (2, 8, 32, 64). See the Usage section for a description of the return codes.

Usage

tran_dumpable_status allows you to determine if dump transaction is allowed on a database without having to run the command. tran_dumpable_status performs all of the checks that Adaptive Server performs when dump transaction is issued.

If tran_dumpable_status returns 0, you can perform the dump transaction command on the database. If it returns any other value, it cannot. The non-0 values are:

- 1 – A database with the name you specified does not exist.
- 2 – A log does not exist on a separate device.
- 4 – The log first page is in the bounds of a data-only disk fragment.
- 8 – the trunc log on chkpt option is set for the database.
- 16 – Non-logged writes have occurred on the database.
- 32 – Truncate-only dump tran has interrupted any coherent sequence of dumps to dump devices.
- 64 – Database is newly created or upgraded. Transaction log may not be dumped until a dump database has been performed.
- 128 – Database durability does not allow transaction dumps.
- 256 – Database is read-only. dump transaction started a transaction, which is not allowed on read-only databases.

- 512 – Database is online for standby access. dump transaction started a transaction, which is not allowed on databases in standby access because the transactoin would disturb the load sequence.
- 1024 – Database is an archive database, which do not support dump transaction.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute this function.

See also

Command dump transaction

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

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, tsequal updates the row. If the values are not equal, tsequal returns this error message:

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp, 0x0001000000002ea8)
```

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 77.
- 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
```

- Do not use tsequal 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 you use a timestamp column as a search clause, compare it 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 using alter table. For example, to add a timestamp column with a NULL value to each existing row:

```
alter table oldtable add timestamp
```

To generate a timestamp, update each existing row without specifying new column values:

```
update oldtable
set col1 = col1
```

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute tsequal.

See also — **Datatype** 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, extract a substring of at least 2 Unicode values (*substr* does not extract half of a surrogate pair).
- If *uchar_expr* is NULL, uhighsurr returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute uhighsurr.

See also

For general information about string functions, see "String functions" on page 76.

Function 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, ulowsurr returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute ulowsurr.

See also

For general information about string functions, see “String functions” on page 76.

Function 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, upper returns NULL.
- Characters that have no upper-ase 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.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute upper.

See also
For general information about string functions, see “String functions” on page 76.

Function lower

uscalar

Description

Returns the Unicode scalar value for the first Unicode character in an expression.

Syntax

uscalar(*uchar_expr*)

Parameters

uchar_expr

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

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.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute uscalar.

See also

For general information about string functions, see "String functions" on page 76.

Functions ascii

used_pages

Description

Reports the number of pages used by a table, an index, or a specific partition. Unlike data_pages, used_pages does include pages used for internal structures. This function replaces the used_pgs function used in versions of Adaptive Server earlier than 15.0.

Syntax

used_pages(*dbid*, *object_id*[, *indid*[, *ptnid*]])

Parameters

dbid
is the database id where target object resides.

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.

indid
is the index id of interest.

ptnid
is the partition id of interest.

Examples

Example 1 Returns the number of pages used by the object with a object ID of 31000114 in the specified database (including any indexes):

```
select used_pages(5, 31000114)
```

Example 2 Returns the number of pages used by the object in the data layer, regardless of whether or not a clustered index exists:

```
select used_pages(5, 31000114, 0)
```

Example 3 Returns the number of pages used by the object in the index layer for an index with index ID 2. This does not include the pages used by the data layer (See the first bullet in the Usage section for an exception):

```
select used_pages(5, 31000114, 2)
```

Example 4 Returns the number of pages used by the object in the data layer of the specific partition, which in this case is 2323242432:

```
select used_pages(5, 31000114, 0, 2323242432)
```

Usage

- In an all-pages locked table with a clustered index, the value of the last parameter determines which pages used are returned:
  - used_pages(dbid, objid, 0) – which explicitly passes 0 as the index ID, returns only the pages used by the data layer.

- used_pages(dbid, objid, 1) – returns the pages used by the index layer as well as the pages used by the data layer.

  To obtain the index layer used pages for an all-pages locked table with a clustered index, subtract used_pages(dbid, objid, 0) from used_pages(dbid, objid, 1).

- In in an all-pages-locked table with a clustered index, used_pages is passed only the used pages in the data layer, for a value of `indid = 0`. When `indid=1` is passed, the used pages at the data layer and at the clustered index layer are returned, as in previous versions.
- used_pages is similar to the old used_pgs(objid, doampg, ioampg) function.
- All erroneous conditions result in a return value of zero.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute used_pgs.

See also **Functions** data_pages, 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".

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute user.

See also
For general information about system functions, see "System functions" on page 77.

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 77.
- 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

ANSI SQL – Compliance 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.
- 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

ANSI SQL – Compliance 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

For general information about system functions, see "System functions" on page 77.

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, and can be up to 255 bytes in length.

Syntax

valid_name(*character_expression*[, *maximum_length*])

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.

maximum_length
is an integer larger than 0 and less than or equal to 255. The default value is 30. If the identifier length is larger than the second argument, valid_name returns 0, and returns a value greater than zero if the identifier length is invalid.

Examples

Creates a procedure to verify that identifiers are valid:

```
create procedure chkname
@name varchar(30)
as
     if valid_name(@name) = 0
     print "name not 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 16384 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 (@).

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute valid_name.

See also

For general information about system functions, see "System functions" on page 77.

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.

Standards

ANSI SQL – Compliance 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

For general information about system functions, see “System functions” on page 77.

System procedures sp_addlogin, sp_adduser

var

Description

Computes the statistical variance of a sample consisting of a numeric expression, as a double, and returns the variance of a set of numbers.

Note var and variance are aliases of var_samp. See var_samp on page 329 for details.

---

var_pop

Description

Computes the statistical variance of a population consisting of a numeric expression, as a double. varp is an alias for var_pop, and uses the same syntax.

Syntax

var_pop ([all | distinct] *expression*)

Parameters

all
applies var_pop to all values. all is the default.

distinct
eliminates duplicate values before var_pop is applied.

expression
is an expression—commonly a column name—in which its population-based variance is calculated over a set of rows.

Examples

Lists the average and variance of the advances for each type of book in the pubs2 database:

```
select type, avg(advance) as "avg", var_pop(advance)
  as "variance" from titles group by type order by type
```

Usage

Computes the population variance of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression, from the mean of value expression, divided by the number of rows in the group or partition.

Figure 2-5: The formula for population-related statistical aggregate functions

The formula that defines the variance of the population of size *n* having mean μ (var_pop) is as follows. The population standard deviation (stddev_pop) is the positive square root of this.

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{n}$$

σ^2 = Variance

n = Population size

μ = Mean of the values x_i

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute var_pop.

See also

For general information about aggregate functions, see “Aggregate functions” in *Adaptive Server Enterprise Reference Manual: Building Blocks*.

Functions stddev_pop, stddev_samp, var_samp

var_samp

Description

Computes the statistical variance of a sample consisting of a numeric-expression, as a double, and returns the variance of a set of numbers. var and variance are aliases of var_samp, and use the same syntax.

Syntax

var_samp ([all | distinct] *expression*)

Parameters

all
applies var_samp to all values. all is the default.

distinct
eliminates duplicate values before var_samp is applied.

expression
is any numeric datatype (float, real, or double) expression.

Examples

Lists the average and variance of the advances for each type of book in the pubs2 database:

```
select type, avg(advance) as "avg", var_samp(advance)
  as "variance" from titles where
  total_sales > 2000 group by type order by type
```

Usage

var_samp returns a result of double-precision floating-point datatype. If applied to the empty set, the result is NULL.

Figure 2-6: The formula for sample-related statistical aggregate functions

The formula that defines an unbiased estimate of the population variance from a sample of size *n* having mean $\bar{x}$ (var_samp) is as follows. The sample standard deviation (stddev_samp) is the positive square root of this.

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

s^2 = Variance

n = Sample size

$\bar{x}$ = Mean of the values x_i

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute var_samp.

See also

For general information about aggregate functions, see “Aggregate functions” in *Adaptive Server Enterprise Reference Manual: Building Blocks*.

Functions stddev_pop, stddev_samp, var_pop

variance

Description

Computes the statistical variance of a sample consisting of a numeric expression, as a double, and returns the variance of a set of numbers.

Note var and variance are aliases of var_samp. See var_samp on page 329 for details.

---

varp

Description

Computes the statistical variance of a population consisting of a numeric expression, as a double.

Note varp is an alias of var_pop. See var_pop on page 327 for details.

workload_metric

Description

Cluster environments only – Queries the current workload metric for the instance you specify, or updates the metric for the instance you specify.

Syntax

workload_metric(*instance_id* | *instance_name* [, *new_value*])

Parameters

instance_id
ID of the instance.

instance_name
name of the instance.

new_value
float value representing the new metric.

Examples

Example 1 Sees the user metric on the current instance:

```
select workload_metric()
```

Example 2 Sees the user metric on instance "ase2":

```
select workload_metric("ase2")
```

Example 3 Sets the value of the user metric on "ase3" to 27.54:

```
select workload_metric("ase3", 27.54)
```

Usage

- A NULL value indicates the current instance.
- If a value is specified for *new_value*, the specified value becomes the current user metric. If a value is not specified for *new_value*, the current workload metric is returned.
- The value of *new_value* must be zero or greater.
- If a value is supplied for *new_value*, workload_metric returns that value if the operation is successful. Otherwise, workload_metric returns -1.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

You must have the sa_role or ha_role to execute workload_metric

xa_bqual

Description

Returns the binary version of the bqual component of an ASCII XA transaction ID.

Syntax

xa_bqual(*xid*, 0)

Parameters

xid
is the ID of an Adaptive Server transaction, obtained from the xactname column in systransactions or from sp_transactions.

0
is reserved for future use

Examples

Example 1 Returns “0x227f06ca80”, the binary translation of the branch qualifier for the Adaptive Server transaction ID “0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0”. The Adaptive Server transaction ID is first obtained using sp_transactions:

```
1> sp_transactions

xactkey                        type     coordinator starttime          st
ate      connection dbid  spid  loid  failover    srvname  namelen  xactna
me
------------------------------ -------- ----------- ------------------ --
-------- ---------- ----- ----- ----- ---------- -------  -------  ------
---------------------------------
0x531600000600000017e4885b0700 External XA          Dec 9 2005  5:15PM In
Command  Attached     7   20    877  Resident Tx  NULL         39
0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0

1> select xa_bqual("0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0", 0)
2> go

...

--------------------------------------------------------------------

0x227f06ca80
```

Example 2 xa_bqual is often used together with xa_gtrid. This example returns the global transaction IDs and branch qualifiers from all rows in systransactions where its coordinator column is the value of “3”:

```
1> select gtrid=xa_gtrid(xactname,0),
    bqual=xa_bqual(xactname,0)
    from systransactions where coordinator = 3
2> go

        gtrid
```

```
        bqual
   ---------------------------------------------------------------
   ---------------------------------------------------------------
         0xb1946cdc52464a61cba42fe4e0f5232b
         0x227f06ca80
```

Usage

If an external transaction is blocked on Adaptive Server and you are using sp_lock and sp_transactions to identify the blocking transaction, you can use the XA transaction manager to terminate the global transaction. However, when you execute sp_transactions, the value of *xactname* it returns is in ASCII string format, while XA Server uses an undecoded binary value. Using xa_bqual thus allows you to determine the bqual portion of the transaction name in a format that can be understood by the XA transaction manager.

xa_bqual returns:

- The translated version of this string that follows the second "_" (underscore) and preceeds either the third "_" or end-of-string value, whichever comes first.
- NULL if the transaction ID cannot be decoded, or is in an unexpected format.

Note xa_bqual does not perform a validation check on the xid, but only returns a translated string.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can use xa_bqual.

See also

Functions xa_gtrid

Stored procedures sp_lock, sp_transactions

xa_gtrid

Description

Returns the binary version of the gtrid component of an ASCII XA transaction ID.

Syntax

xa_gtrid(*xactname*, *int*)

Parameters

xid
is the ID of an Adaptive Server transaction, obtained from the xactname column in systransactions or from sp_transactions.

0
is reserved for future use

Examples

Example 1 In this typical situation, returns “0x227f06ca80,” the binary translation of the branch qualifier, and “0xb1946cdc52464a61cba42fe4e0f5232b,” the global transaction ID, for the Adaptive Server transaction ID “0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0”:

```
1> select xa_gtrid("0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0", 0)
2> go

        ...

--------------------------------------------------------------------------
          0xb1946cdc52464a61cba42fe4e0f5232b

(1 row affected)
```

Example 2 xa_bqual is often used together with xa_gtrid. This example returns the global transaction IDs and branch qualifiers from all rows in systransactions where its coordinator column is the value of “3”:

```
1> select gtrid=xa_gtrid(xactname,0),
    bqual=xa_bqual(xactname,0)
    from systransactions where coordinator = 3
2> go
        gtrid

        bqual

--------------------------------------------------------------------------

--------------------------------------------------------------------------

         0xb1946cdc52464a61cba42fe4e0f5232b
```

```
0x227f06ca80
```

Usage

If an external transaction is blocked on Adaptive Server and you are using sp_lock and sp_transactions to identify the blocking transaction, you can use the XA transaction manager to terminate the global transaction. However, when you execute sp_transactions, the value of *xactname* it returns is in ASCII string format, while XA Server uses an undecoded binary value. Using xa_gtrid thus allows you to determine the gtrid portion of the transaction name in a format that can be understood by the XA transaction manager.

xa_gtrid returns:

- The translation version of tis string that follows the first "_" (underscore) and preceeds either the second "_" or end-of-string value, whichever comes first.
- NULL if the transaction ID cannot be decoded, or is in an unexpected format.

Note xa_gtrid does not perform a validation check on the xid, but only returns a translated string.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can use xa_gtrid.

See also

Functions xa_bqual

Stored procedures sp_lock, sp_transactions

xmltable

Description

Extracts data from an XML document and returns it as a SQL table.

Syntax

```
xmltable_expression ::= xmltable
            ( row_pattern passing xml_argument
            columns column_definitions
            options_parameter)
        row_pattern ::= character_string_literal
        xml_argument ::=  xml_expression | column_reference |
variable_reference
        column_definitions ::=
        column_definition [ { , column_definition } ]
                column_definition ::=
            ordinality_column | regular_column

ordinality_column ::= column_name datatype for ordinality
        regular_column ::=
            column_name datatype [ default literal ] [null | not null]
        [ path column_pattern ]
        column_pattern ::= character_string_literal
        options_parameter ::=[,] option option_string
        options_string ::= basic_string_expression
```

Derived table syntax Returns a SQL table from within a SQL from clause.

```
    from_clause ::= from table_reference [, table_reference]...
    table_reference ::= table_view_name | ANSI_join |derived_table
    table_view_name::=See the select command in Reference Manual
            Volume 2, "Commands."
    ANSI_join::=See the select command in Reference Manual
            Volume 2, "Commands."
    derived_table ::=
        (subquery) as table_name [ (column_name [, column_name]...)|
        xmltable_expression as table_name
```

Parameters

xml_argument
: is an expression, column reference, or variable, referring to an XML document.

for
: is a reserved XML keyword.

ordinality
: is a non-reserved XML keyword.

passing
: is a non-reserved XML keyword.

row_pattern
is an XPath query expression whose result is a sequence of elements from the specified document. The xmltable call returns a table with one row for each element in the sequence.

columns
is a non-reserved XML keyword.

column_name
is the user-specified name of the column.

column_pattern
is an XPath query expression that applies to an element of the sequence returned by the *row_pattern*, to extract the data for a column of the result table. If the *column_pattern* is omitted, the *column_pattern* defaults to the *column_name*.

ordinality_column
is a column of datatypes integer, smallint, tinyint, decimal, or numeric, which indicates ordering of the elements in the input XML document.

regular_column
is any column that is not an ordinality column.

derived_table
is a parenthesized subquery specified in the from clause of a SQL query.

path
is a reserved XML keyword.

option
is an *option_string*, defined in *XML Services*, and a reserved XML keyword.

Examples

Example 1 Shows a simple xmltable call with the document specified as a character-string literal:

```
select * from xmltable('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'
  columns id int path 'id',name varchar(20) path 'name') as items_table
id              name
-----           -----
1               Box
2               Jar

(2 rows affected)
```

Example 2 Stores the document in a Transact-SQL variable, and references that variable in the xmltable call:

```
declare @doc varchar(16384)
set @doc='<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'

select * from xmltable('/doc/item' passing @doc
  columns id int path 'id', name varchar(20) path 'name') as items_table

id              name
----            -----
1               Box
2               Jar

(2 rows affected)
```

Example 3 Stores the document in a table and references it with a subquery:

```
select 100 as doc_id,
'<doc><item><id>1</id><name>Box</name></item><item><id>2</id>
  <name>Jar</name></item></doc>' as doc
into #sample_docs
select * from xmltable('/doc/item'
  passing(select doc from #sample_docs where doc_id=100)
  columns id int path 'id',name varchar(20) path 'name') as items_table

id              name
-------         -----
1               Box
2               Jar

(2 rows affected)
```

Example 4 If a row pattern returns an empty sequence, the result is an empty table:

```
select * from xmltable ('/doc/item_entry'
  passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'
  columns id int path 'id',
    name varchar(20) path 'name') as items_table

id                 name
-------          ------

(0 rows affected)
```

Example 5 The arguments following the columns keyword comprise the list of column definitions. Each column definition specifies a column name and datatype, as in create table, and a path, called the column pattern.

When the data for a column is contained in an XML attribute, specify the column pattern using “@” to reference an attribute. For example:

```
select * from xmltable ('/doc/item'
  passing '<doc><item id="1"><name>Box</name></item>'
    +'<item id="2"><name>Jar</name></item></doc>'
  columns id int path '@id', name varchar(20)) as items_table

id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

Example 6 A *column-pattern* is commonly the same as the specified *column_name*, for example name. In this case, omitting the column-pattern results in defaulting to the *column_name*:

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'
  columns id int, name varchar(20)) as items_table

id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

Example 7 If you want a column pattern to default to the column name, in a column whose value is in an XML attribute, use a quoted identifier. You must then quote such identifiers when you reference them in the results:

```
set quoted_identifier on
select "@id", name from xmltable ('/doc/item'
   passing '<doc><item id="1"><name>Box</name></item>'
      +'<item id="2"><name>Jar</name></item></doc>'
   columns "@id" int, name varchar(20)) as items_table

@id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

Example 8 You can also use quoted identifiers to specify column names as default column patterns, using column names that are more complex XPath expressions. For example:

```
set quoted_identifier on
select "@id", "name/short", "name/full" from xmltable ('/doc/item'
  passing '<doc><item id="1"><name><short>Box</short>
      <full>Box, packing, moisture resistant, plain</full>
      </name></item>'
  +'<item id="2"><name><short>Jar</short>
      <full>Jar, lidded, heavy duty</full>
      </name></item></doc>'
  columns "@id" int, "name/short" varchar(20), "name/full" varchar(50))
  as items_table

@id               name/short    name/full
--------------------
1                Box           Box, packing, moisture resistant, plain
2                Jar           Jar, lidded, heavy duty

(2 rows affected)
```

Example 9 The function text is implicit in column patterns. This example does not specify text in the column pattern for either the id or name column:

```
select * from xmltable ('/doc/item'
passing '<doc><item><id>1</id><name>Box</name></item>'
+ '<item><id>2</id><name>Jar</name></item></doc>'
columns id int path 'id', name varchar (20) path 'name') as items_table
id    name
---   --------
1     Box
2     Jar
(2 rows affected)
```

Example 10 Applying an implicit SQL converst statement to the data extracted from the column pattern, derives column values in datatype conversions.

```
select * from xmltable ('/emps/emp'
  passing '<emps>
<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp>'
+'<emp><id>2</id><salary>234.56</salary><hired>2/3/2004</hired></emp>'
    +'</emps>'
  columns id int path 'id', salary dec(5,2), hired date)
as items_table

id         salary        hired
-------    ---------     --------
```

```
1          123.45       Jan 2, 2003
2          234.56       Feb 3, 2004
(2 rows affected)
```

Example 11 You can use an *ordinality_column* in xmltable to record the ordering of elements in the input XML document:

```
declare @doc varchar(16384)
set @doc = '<doc><item><id>25</id><name>Box</name></item>'
  +'<item><id>15</id><name>Jar</name></item></doc>'
select * from xmltable('/doc/item' passing @doc
  columns item_order int for ordinality,
    id int path 'id',
    name varchar(20) path 'name') as items_table
order by item_order

item_order         id            name
----------         ---           ----
1                  25            Box
2                  15            Jar
(2 rows affected)
```

Without the for ordinality clause and the item_order column, there is nothing in the id and name columns that indicates that the row of id 25 precedes the row of id 15. The for ordinality clause orders the output SQL rows the same as the ordering of the elements in the input XML document.

The datatype of an ordinality column can be any fixed numeric datatype: int, tinyint, bigint, numeric, or decimal. numeric and decimal must have a scale of 0. An ordinality column cannot be real or float.

Example 12 Omits the <name> element from the second <item>. The name column allows names to be NULL by default.

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) path 'name')
as items_table
id            name
------------------
1             Box
2             NULL

(2 rows affected)
```

Example 13 Omits the <name> element from the second <item>, and specifies not null for the name column:

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) not null path 'name')
as items_table

id              name
-----           --------
1               Box
Msg 14847, Level 16, State 1:
Line 1:
XMLTABLE column 0, does not allow null values.
```

Example 14 Adds a default clause to the name column, and omits the <name> elements from the second <item>.

```
select * from xmltable ('/doc/item'
  passing'<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) default '***' path 'name')
as items_table
id   name
---  ------
1    Box
2    '***'
(2 rows affected)
```

Example 15 Shows SQL commands in which you can use an xmltable call in a derived table expression. This example uses xmltable in a simple select statement:

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id><name>Jar</name></item></doc>'
columns id int path 'id',
  name varchar(20) path 'name') as items_table

id             name
--             ----
1              Box
2              Jar
(2 rows affected)
```

Example 16 Uses xmltable in a view definition. It stores a document in a table and references that stored document in a create view statement, using xmltable to extract data from the table:

```
select 100 as doc_id,
```

```
'<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id><name>Jar</name></item></doc>' as doc
into sample_docs
create view items_table as
  select * from xmltable('/doc/item'
    passing (select doc from sample_docs where doc_id=100)
    columns id int path 'id',
    name varchar(20) path 'name') as xml_extract
select * from items_table

id             name
------------------
1              Box
2              Jar
(2 rows affected)
```

Example 17 Uses xmltable in a cursor:

```
declare C cursor for
select * from xmltable ('/doc/item'
  passing (select doc from sample_docs where id=100)
  columns id int path 'id',
  name varchar(20) path 'name') as items_table
  go
declare @idvar int
declare @namevar varchar(20)
open C
while @@sqlstatus=0
begin
fetch C into @idvar, @namevar
print 'ID "%1!" NAME"%2!"', @idvar, @namevar
end
-------------------------
ID "1" NAME "Box"
ID "2" NAME "Jar"

(2 rows affected)
```

In applications that require multiple actions for each generated row, such as executing update, insert, or delete from other tables, you can process an xmltable result with a cursor loop. Alternatively, store the xmltable result in a temporary table and process that table with a cursor loop.

Example 18 This example uses xmltable in select into:

```
select * into #extracted_table
from xmltable('/doc/item'
  passing (select doc from sample_docs where doc_id=100
```

```
  columns id int path 'id',
  name varchar(20) path 'name') as items_table

select * from #extracted_table

id            name
----          ------------
1             Box
2             Jar
```

Example 19 Uses xmltable in an insert command:

```
create table #extracted_data (idcol int, namecol varchar(20))
insert into #extracted_data
select * from xmltable('/doc/item'
  passing (select doc from sample_docs where doc_id=100)
  columns id int path 'id', name varchar(20) path 'name') as items_table
select * from #extracted_data

idcol          namecol
-----          ---------
1              Box
2              Jar
(2 rows affected)
```

Example 20 Uses xmltable in a subquery. xmltable returns a SQL table, so the subquery must perform either an aggregation or a selection to return a single row and column for the subquery result.

```
declare @idvar int
set @idvar = 2
select @idvar,
(select name from xmltable ('/doc/item'
  passing(select doc from sample_docs where doc_id=100)
  columns id int path 'id',name varchar(20) path 'name') as item_table
where items_table.id=@idvar)
----------------------------
2                   Jar
(1 rows affected)
```

Example 21 Joins an xmltable result with other tables, using either multiple table joins in the from clause, or outer joins:

```
create table prices (id int, price decimal (5,2))
insert into prices values(1,123.45)
insert into prices values (2,234.56)
select prices.id,extracted_table.name, prices.price
from prices,(select * from xmltable('/doc/item'
  passing (select doc from sample_docs where doc_id=100)
```

```
  columns id int path 'id', name varchar(20) path 'name') as a) as
extracted_table
where prices.id=extracted_table.id
id name  price
-- ----  -----
1  Box   123.45
2  Jar   234.56
(2 rows affected)
```

Example 22 Uses xmltable, with a lateral reference to a column existing in a preceding table in the same from clause as xmltable:

```
create table deptab (col1 int, col2 image)
insert deptab values (1, '<dept>
                          <dept-id>1</dept-id>
                          <dept-name>Finance</dept-name>
                          <employees>
                            <emp><name>John</name><id>e11</id></emp>
                            <emp><name>Bela</name><id>e12</id></emp>
                            <emp><name>James</name><id>e13</id></emp>
                          </employees>
                          </dept>')

insert deptab values (2, '<dept>
                          <dept-id>2</dept-id>
                          <dept-name>Engineering</dept-name>
                          <employees>
                            <emp><name>Tom</name><id>e21</id></emp>
                            <emp><name>Jeff</name><id>e22</id></emp>
                            <emp><name>Mary</name><id>e23</id></emp>
                          </employees>
                          </dept>')

select id, empname from deptab, xmltable ('/dept/employees/emp' passing
deptab.co12 columns empname varchar (8) path 'name', id varchar (8)
path 'id') as sample_tab

id                    empname
---------             -----------
e11                   John
e12                   Bela
e13                   James
e21                   Tom
e22                   Jeff
e23                   Mary
(6 rows affected)
```

Usage

- xmltable is a built-in, table-valued function.

- The syntax of derived tables requires you to specify a table name, even if you do not reference it. Therefore, each xmltable expression must also specify a table name.
- The argument following passing is the input XML document.
- The result type of an xmltable expression is a SQL table, whose column names and their datatypes are specified by *column_definitions*.
- To process documents, you can apply xmltable to the XML document in each row of a table of XML documents.
- These keywords are associated with xmltable:
  - Reserved – for, option, xmltable, path
  - Not reserved – columns, ordinality, passing
- The expressions in the arguments of an xmltable call can reference the column names of preceding tables in the from clause containing the xmltable call. Only tables that precede the xmltable call can be referenced. Such a reference, to a column of a preceding table in the same from clause, is called a **lateral reference**. For example:

```
select * from T1, xmltable(...passing T1.C1...)
       as XT2, xmltable(...passing XT2.C2...)as XT3
```

  The reference to T1.C1 in the first xmltable call is a lateral reference to column C1 of table T1. The reference to XT2.C2 in the second xmltable call is a lateral reference to column C2 of the table generated by the first xmltable call.

- You cannot use xmltable in the from clause of an update or delete statement. For example, the following statement fails:

```
update T set T.C=...
from xmltable(...) as T
where...
```

- Datatypes in *regular_columns* can be of any SQL datatype.
- To handle XML data whose format is not suitable for a SQL convert function, extract the data to a string column (varchar, text, image, java.lang.String).
- The extracted XML data for the column must be convertible to the column datatype, or an exception is raised.
- If a column pattern returns an empty result, the action taken depends on the `default` and `{null | not null}` clauses.

- The literal following a default in a *regular_column* must be assignable to the datatype of the column.
- There can be no more than one *ordinality_column*; the datatype specified for this variable must be integer, smallint, tinyint, decimal, or numeric. decimal and numeric must have a scale of zero.
- An *ordinality_column*, if one exists, is not nullable.

Note This default is different from the default value of create table.

- The nullable property of other columns is specified by the `{null | not null}` clause. The default is null.
- The current setting of set quoted_identifier applies to the clauses of an xmltable expression. For example:
  - If set quoted_identifier is on, column names can be quoted identifiers, and string literals in *row_pattern*, *column_pattern*, and default literals must be surrounded with single quotation marks.
  - If set quoted_identifier is off, column names cannot be quoted identifiers, and string literals in *row_pattern*, *column_pattern*, and default literals can be surrounded with either single or double quotation marks.
- The general format of the option_string is described in “option_strings: general format,” in *XML Services, Adaptive Server 15.0*.

xmltable row and column patterns

- xmltable row and column patterns are allowed to be only simple paths. Simple paths in XPath consist only of forward traversals using '/' and element/attribute names.
- If the *row_pattern* does not begin at the root level of the document specified by *xml_argument*, an exception is raised. The row pattern must begin at the root of the XML document.
- The row pattern expression cannot contain an XPath function.
- A column pattern must be a relative path.
- If the *row_pattern* specifies an XML function, an exception is raised. The row pattern cannot specify an XML function.
- If a *column_definition* does not specify a path, the default *column_pattern* is the *column_name* of the column definition. This default is subject to the case sensitivity of the server. For example, consider this statement:

```
select * from xmltable(...columns name
   varchar(30),...)
```

If the server is case-insensitive, this is equivalent to the following:

```
select * from xmltable(...columns name varchar(30)
path 'name',...)
```

If the server is case-sensitive, the first statement is equivalent to:

```
select * from xmltable
   (...columns name varchar(30)path 'NAME',...)
```

Generating the rows of the result table

- The result value of an xmltable expression is a T-SQL table RT, defined as follows:
- RT has a row for each element in the XML sequence that results from applying the *row_pattern* to the *xml_argument*.
- The rows of RT have a column for each *column_definition*, with the *column_name* and datatype specified in the *column_definition*.
- If a *column_definition* is a *ordinality_column*, its value for the Nth row is the integer N.
- If a *column_definition* is a *regular_column*, its value for the Nth row corresponds to the following:
  - Let XVAL be the result of applying this XPath expression to the *xml_argument*:

    ```
    (row_pattern[N])/column_pattern/text()
    ```

  - If XVAL is empty, and the *column_definition* contains a default clause, the value of the column is that default value.

    If XVAL is empty and the *column_definition* specifies not null, an exception is raised.

    Otherwise, the value of the column is the null value.
  - If XVAL is not empty, and the datatype of the column is char, varchar, text, unitext, unichar, univarchar, or java.lang.String, de-entitize XVAL.
  - The value of the column is the result of:

    ```
    convert(datatype,XVAL)
    ```

year

Description — Returns an integer that represents the year in the datepart of a specified date.

Syntax — year(*date_expression*)

Parameters — *date_expression*

is an expression of type datetime, smalldatetime, date, time or a character string in a datetime format.

Examples — Returns the integer 03:

```
year("11/02/03")
----------
03
(1 row(s) affected)
```

Usage — year(*date_expression*) is equivalent to datepart(*yy*, *date_expression*).

Standards — ANSI SQL – Compliance level: Transact-SQL extension.

Permissions — Any user can execute year.

See also — **Datatypes** datetime, smalldatetime, date

Functions datepart, day, month

CHAPTER 3 **Global Variables**

Topics	Page

Adaptive Server global variables

Global variables are system-defined variables updated by Adaptive Server while the system is running. Some global variables are session-specific, while others are server instance-specific. For example, *@@error* contains the last error number generated by the system for a given user connection.

See get_appcontext and set_appcontext to specify application context variables.

To view the value for any global variable, enter:

select *variable_name*

For example:

```
select @@char_convert
```

Table 3-1lists the global variables available for Adaptive Server:

Table 3-1: Adaptive Server global variables

Global variable	Definition
@@active_instances	Returns the number of active instances in the cluster
@@authmech	A read-only variable that indicates the mechanism used to authenticate the user.
@@bootcount	Returns the number of times an Adaptive Server installation has been booted.
@@boottime	Returns the date and time Adaptive Server was last booted.
@@bulkarraysize	Returns the number of rows to be buffered in local server memory before being transferred using the bulk copy interface Used only with Component Integration Services for transferring rows to a remote server using select into. For more information, see the *Component Integration Services User's Guide*.

Global variable	Definition
@@bulkbatchsize	Returns the number of rows transferred to a remote server via select into *proxy_table* using the bulk interface. Used only with Component Integration Services for transferring rows to a remote server using select into. For more information, see the *Component Integration Services User's Guide*.
@@char_convert	Returns 0 if character set conversion is not in effect. Returns 1 if character set conversion is in effect.
@@cis_rpc_handling	Returns 0 if cis rpc handling is off. Returns 1 if cis rpc handling is on. For more information, see the *Component Integration Services User's Guide*.
@@cis_version	Returns the date and version of Component Integration Services.
@@client_csexpansion	Returns the expansion factor used when converting from the server character set to the client character set. For example, if it contains a value of 2, a character in the server character set could take up to twice the number of bytes after translation to the client character set.
@@client_csid	Returns -1 if the client character set has never been initialized; returns the client character set ID from syscharsets for the connection if the client character set has been initialized.
@@client_csname	Returns NULL if client character set has never been initialized; returns the name of the character set for the connection if the client character set has been initialized.
@@clusterboottime	Returns the date and time the cluster was first started, even if the instance that originally started the cluster start has shut down
@@clustercoordid	Returns the instance id of the current cluster coordinator
@@clustermode	Returns the string: "shared-disk cluster"
@@clustername	Returns the name of the cluster
@@cmpstate	Returns the current mode of Adaptive Server in a high availability environment. Not used in a non-high availability environment.
@@connections	Returns the number of user logins attempted.
@@cpu_busy	Returns the amount of time, in ticks, that the CPU has spent doing Adaptive Server work since the last time Adaptive Server was started.

Global variable	Definition

@@cursor_rows	A global variable designed specifically for scrollable cursors. Displays the total number of rows in the cursor result set. Returns the following values:
	• -1 – the cursor is:
	  • Dynamic – because dynamic cursors reflect all changes, the number of rows that qualify for the cursor is constantly changing. You can never be certain that all the qualified rows are retrieved.
	  • semi_sensitive and scrollable, but the scrolling worktable is not yet fully populated – the number of rows that qualify the cursor is unknown at the time this value is retrieved.
	• 0 – either no cursors are open, no rows qualify for the last opened cursor, or the last open cursor is closed or deallocated.
	• *n* – the last opened or fetched cursor result set is fully populated. The value returned is the total number of rows in the cursor result set.

@@curloid	Either no cursors are open, no rows qualify for the last opened cursor, or the last open cursor is closed or deallocated.
@@datefirst	Set using set datefirst *n* where *n* is a value between 1 and 7. Returns the current value of *@@datefirst*, indicating the specified first day of each week, expressed as tinyint. The default value in Adaptive Server is Sunday (based on the us_language default), which you set by specifying `set datefirst 7`. See the datefirst option of the set command for more information on settings and values.
@@dbts	Returns the timestamp of the current database.
@@error	Returns the error number most recently generated by the system.
@@errorlog	Returns the full path to the directory in which the Adaptive Server error log is kept, relative to *$SYBASE* directory (*%SYBASE%* on NT).
@@failedoverconn	Returns a value greater than 0 if the connection to the primary companion has failed over and is executing on the secondary companion server. Used only in a high availability environment, and is session-specific.
@@fetch_status	Returns: • 0 – fetch operation successful • -1 – fetch operation unsuccessful • -2 – value reserved for future use
@@guestuserid	Returns the ID of the guest user.
@@hacmpservername	Returns the name of the companion server in a high availability setup.
@@haconnection	Returns a value greater than 0 if the connection has the failover property enabled. This is a session-specific property.
@@heapmemsize	Returns the size of the heap memory pool, in bytes. See the *System Administration Guide* for more information on heap memory.
@@identity	Returns the most recently generated IDENTITY column value.

Global variable	Definition
@@idle	Returns the amount of time, in ticks, that Adaptive Server has been idle since it was last started.
@@instanceid	Returns the id of the instance from which it was executed
@@instancename	Returns the name of the instance from which it was executed
@@invaliduserid	Returns a value of -1 for an invalid user ID.
@@io_busy	Returns the amount of time, in ticks, that Adaptive Server has spent doing input and output operations.
@@isolation	Returns the value of the session-specific isolation level (0, 1, or 3) of the current Transact-SQL program.
@@jsinstanceid	ID of the instance on which the Job Scheduler is running, or will run once enabled.
@@kernel_addr	Returns the starting address of the first shared memory region that contains the kernel region. The result is in the form of 0x*address pointer value*.
@@kernel_size	Returns the size of the kernel region that is part of the first shared memory region.
@@langid	Returns the server-wide language ID of the language in use, as specified in syslanguages.langid.
@@language	Returns the name of the language in use, as specified in syslanguages.name.
@@lastlogindate	Available to each user login session, *@@lastlogindate* includes a datetime datatype, its value is the lastlogindate column for the login account before the current session was established. This variable is specific to each login session and can be used by that session to determine the previous login to the account. If the account has not been used previously or "sp_passwordpolicy 'set', enable last login updates" is 0, then the value of @@lastlogindate is NULL.
@@lock_timeout	Set using set lock wait n. Returns the current *lock_timeout* setting, in milliseconds. *@@lock_timeout* returns the value of n. The default value is no timeout. If no set lock wait n is executed at the beginning of the session, *@@lock_timeout* returns -1.
@@maxcharlen	Returns the maximum length, in bytes, of a character in Adaptive Server's default character set.
@@max_connections	Returns the maximum number of simultaneous connections that can be made with Adaptive Server in the current computer environment. You can configure Adaptive Server for any number of connections less than or equal to the value of *@@max_connections* with the number of user connections configuration parameter.
@@maxgroupid	Returns the highest group user ID. The highest value is 1048576.
@@maxpagesize	Returns the server's logical page size.
@@max_precision	Returns the precision level used by decimal and numeric datatypes set by the server. This value is a fixed constant of 38.
@@maxspid	Returns maximum valid value for the spid.
@@maxsuid	Returns the highest server user ID. The default value is 2147483647.
@@maxuserid	Returns the highest user ID. The highest value is 2147483647.
@@mempool_addr	Returns the global memory pool table address. The result is in the form 0x*address pointer value*. This variable is for internal use.

Global variable	Definition
@@min_poolsize	Returns the minimum size of a named cache pool, in kilobytes. It is calculated based on the DEFAULT_POOL_SIZE, which is 256, and the current value of max database page size.
@@mingroupid	Returns the lowest group user ID. The lowest value is 16384.
@@minspid	Returns 1, which is the lowest value for spid.
@@minsuid	Returns the minimum server user ID. The lowest value is -32768.
@@minuserid	Returns the lowest user ID. The lowest value is -32768.
@@monitors_active	Reduces the number of messages displayed by sp_sysmon.
@@ncharsize	Returns the maximum length, in bytes, of a character set in the current server default character set.
@@nestlevel	Returns the current nesting level.
@@nodeid	Returns the current installation's 48-bit node identifier. Adaptive Server generates a nodeid the first time the master device is first used, and uniquely identifies an Adaptive Server installation.
@@optgoal	Returns the current optimization goal setting for query optimization
@@options	Returns a hexadecimal representation of the session's set options.
@@opttimeoutlimit	Returns the current optimization timeout limit setting for query optimization
@@pack_received	Retruns the number of input packets read by Adaptive Server.
@@pack_sent	Returns the nmber of output packets written by Adaptive Server.
@@packet_errors	Returns the number of errors detected by Adaptive Server while reading and writing packets.
@@pagesize	Returns the server's virtual page size.
@@parallel_degree	Returns the current maximum parallel degree setting.
@@probesuid	Returns a value of 2 for the probe user ID.
@@procid	Returns the stored procedure ID of the currently executing procedure.
@@quorum_physname	Returns the physical path for the quorum device
@@recovery_state	Indicates whether Adaptive Server is in recovery based on these returns: • NOT_IN_RECOVERY – Adaptive Server is not in startup recovery or in failover recovery. Recovery has been completed and all databases that can be online are brought online. • RECOVERY_TUNING – Adaptive Server is in recovery (either startup or failover) and is tuning the optimal number of recovery tasks. • BOOTIME_RECOVERY – Adaptive Server is in startup recovery and has completed tuning the optimal number of tasks. Not all databases have been recovered. • FAILOVER_RECOVER – Adaptive Server is in recovery during an HA failover and has completed tuning the optimal number of recovery tasks. All databases are not brought online yet.
@@repartition_degree	Returns the current dynamic repartitioning degree setting

Global variable	Definition
@@resource_granularity	Returns the maximum resource usage hint setting for query optimization
@@rowcount	Returns the number of rows affected by the last query. The value of *@@rowcount* is affected by whether the specified cursor is forward-only or scrollable. If the cursor is the default, non-scrollable cursor, the value of *@@rowcount* increments one by one, in the forward direction only, until the number of rows in the result set are fetched.These rows are fetched from the underlying tables to the client. The maximum value for *@@ rowcount* is the number of rows in the result set. In the default cursor, *@@rowcount* is set to 0 by any command that does not return or affect rows, such as an if or set command, or an update or delete statement that does not affect any rows. If the cursor is scrollable, there is no maximum value for *@@rowcount*. The value continues to increment with each fetch, regardless of direction, and there is no maximum value. The *@@rowcount* value in scrollable cursors reflects the number of rows fetched from the result set, not from the underlying tables, to the client.
@@scan_parallel_degree	Returns the current maximum parallel degree setting for nonclustered index scans.
@@servername	Returns the name of Adaptive Server.
@@setrowcount	Returns the current value for set rowcount
@@shmem_flags	Returns the shared memory region properties. This variable is for internal use. There are a total of 13 different properties values corresponding to 13 bits in the integer. The valid values represented from low to high bit are: MR_SHARED, MR_SPECIAL, MR_PRIVATE, MR_READABLE, MR_WRITABLE, MR_EXECUTABLE, MR_HWCOHERENCY, MR_SWCOHERENC, MR_EXACT, MR_BEST, MR_NAIL, MR_PSUEDO, MR_ZERO.
@@spid	Returns the server process ID of the current process.
@@sqlstatus	Returns status information (warning exceptions) resulting from the execution of a fetch statement.
@@ssl_ciphersuite	Returns NULL if SSL is not used on the current connection; otherwise, it returns the name of the cipher suite you chose during the SSL handshake on the current connection.
@@stringsize	Returns the amount of character data returned from a toString() method. The default is 50. Max values may be up to 2GB. A value of zero specifies the default value. See the *Component Integration Services User's Guide* for more information.
@@system_busy	Number of ticks during which Adaptive Server was running a system task[1]
@@sys_tempdbid	Returns the database id of the executing instance's effective local system temporary database
@@system_view	Returns the session-specific system view setting, either "instance" or "cluster"
@@tempdbid	Returns a valid temporary database ID (dbid) of the session's assigned temporary database.
@@textcolid	Returns the column ID of the column referenced by *@@textptr*.

Global variable	Definition
@@textdataptnid	Returns the partition ID of a text partition containing the column referenced by @@textptr.
@@textdbid	Returns the database ID of a database containing an object with the column referenced by *@@textptr*.
@@textobjid	Returns the object ID of an object containing the column referenced by *@@textptr*.
@@textptnid	Returns the partition ID of a data partition containing the column referenced by @@textptr.
@@textptr	Returns the text pointer of the last text, unitext, or image column inserted or updated by a process (Not the same as the textptr function).
@@textptr_parameters	Returns 0 if the current status of the textptr_parameters configuration parameter is off. Returns 1 if the current status of the textptr_parameters if on. See the *Component Integration Services User's Guide* for more information.
@@textsize	Returns the limit on the number of bytes of text, unitext, or image data a select returns. Default limit is 32K bytes for isql; the default depends on the client software. Can be changed for a session with set textsize.
@@textts	Returns the text timestamp of the column referenced by *@@textptr*.
@@thresh_hysteresis	Returns the decrease in free space required to activate a threshold. This amount, also known as the hysteresis value, is measured in 2K database pages. It determines how closely thresholds can be placed on a database segment.
@@timeticks	Returns the number of microseconds per tick. The amount of time per tick is machine-dependent.
@@total_errors	Returns the number of errors detected by Adaptive Server while reading and writing.
@@total_read	Returns the number of disk reads by Adaptive Server.
@@total_write	Returns the number of disk writes by Adaptive Server.
@@tranchained	Returns 0 if the current transaction mode of the Transact-SQL program is unchained. Returns 1 if the current transaction mode of the Transact-SQL program is chained.
@@trancount	Returns the nesting level of transactions in the current user session.
@@transactional_rpc	Returns 0 if RPCs to remote servers are transactional. Returns 1 if RPCs to remote servers are not transactional. For more information, see enable xact coordination and set option transactional_rpc in the *Reference Manual*. Also, see the *Component Integration Services User's Guide*.
@@transtate	Returns the current state of a transaction after a statement executes in the current user session.
@@unicharsize	Returns 2, the size of a character in unichar.
@@user_busy	Number of ticks during which Adaptive Server was running a user task[1]
@@version	Returns the date, version string, and so on of the current release of Adaptive Server.
@@version_number	Returns the whole version of the current release of Adaptive Server as an integer
@@version_as_integer	Returns the number of the last upgrade version of the current release of Adaptive Server as an integer. For example, *@@version_as_integer* returns 12500 if you are running Adaptive Server version 12.5, 12.5.0.3, or 12.5.1.

[1]The value of *@@user_busy* + *@@system_busy* should equal the value of *@@cpu_busy*

Using global variables in a clustered environment

For *@@servername*, the Cluster Edition returns the name of the cluster, not the instance name. Use *@@instancename* to return the name of the instance.

In a non-clustered Adaptive Server environment, the value for *@@identity* changes for every record inserted. If the most recent record inserted contains a column with the IDENTITY property, *@@identity* is set to the value of this column, otherwise it is set to "0" (an invalid value). This variable is session-specific, and takes its value based on the last insert that occurred during this session.

In a clustered environment, multiple nodes perform inserts on tables, so the session-specific behavior is not retained for *@@identity*. In a clustered environment, the value for *@@identity* depends on the last record inserted in the node for the current session and not on the last record inserted in the cluster.

CHAPTER 4 Expressions, Identifiers, and Wildcard Characters

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

Topics covered are:

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 4-1 lists the types of expressions that are used in Adaptive Server syntax statements.

Table 4-1: 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

Size of expressions

Expressions returning binary or character datum can be up to 16384 bytes in length. However, earlier versions of Adaptive Server only allowed expressions to be up to 255 bytes in length. If you have upgraded from an earlier release of Adaptive Server, and your stored procedures or scripts store a result string of up to 255 bytes, the remainder will be truncated. You may have to re-write these stored procedures and scripts for to account for the additional length of the expressions.

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 4-2: 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, numeric, 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 4-3 summarizes the results for operands of 0 and 1. If either operand is NULL, the bitwise operator returns NULL:

Table 4-3: 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 4-4 use two tinyint arguments, A = 170 (10101010 in binary form) and B = 75 (01001011 in binary form).

Table 4-4: 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 1s are changed to 0s and all 0s to 1s

String concatenation operator

You can use both the + and || (double-pipe) string operators to concatenate two or more character or binary expressions. For example, the following 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 Name = (au_lname + ", " + au_fname)
    from authors
```

This example results in `"abcdef", "abcdef"`:

```
select "abc" + "def", "abc" || "def"
```

The following 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:

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

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 4-5:

Table 4-5: Comparison operators

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

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 in *Reference Manual: Commands*.

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 following range is inclusive:

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

The following range is not inclusive:

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

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, the following evaluates to NULL if *column1* is NULL:

```
1 + column1
```

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 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 is equivalent to "abc def", not to "abcdef":

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

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 4-6 shows the results of logical operations, including those that involve null values.

Table 4-6: 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 365 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 following example is stored as "abc def":

```
"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.

The limit for the length of object names or identifiers is 255 bytes for regular identifiers, and 253 bytes for delimited identifiers. The limit applies to most user-defined identifiers including table name, column name, index name and so on. Due to the expanded limits, some system tables (catalogs) and built-in functions have been expanded.

For variables, “@” count as 1 byte, and the allowed name for it is 254 bytes long.

Listed below are the identifiers, system tables, and built-in functions that are affected these limits.

The maximum length for these identifiers is now 255 bytes.

- Table name
- Column name
- Index name
- View name
- User-defined datatype
- Trigger name
- Default name
- Rule name
- Constraint name
- Procedure name
- Variable name
- JAR name
- Name of LWP or dynamic statement
- Function name
- Name of the time range
- Application context name

Most user-defined Adaptive Server identifiers can be a maximum of 255 bytes in length, whether single-byte or multibyte characters are used. Others can be a mximum of 30 bytes. Refer to the *Transact-SQL User’s Guide* for a list of both 255-byte and 30-byte identifiers.

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 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 5, “Reserved Words.”

You cannot use the dash symbol (–) as an identifier.

Short identifiers

The maximum length for these identifiers is 30 bytes:

- Cursor name
- Server name
- Host name
- Login name
- Password
- Host process identification
- Application name
- Initial language name
- Character set name
- User name
- Group name
- Database name
- Logical device name
- Segment name

- Session name
- Execution class name
- Engine name
- Quiesce tag name
- Cache name

Tables beginning with # (temporary tables)

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

Adaptive Server performs special operations on temporary table names to maintain unique naming on a per-session basis. When you create a temporary table with a name of fewer than 238 bytes, the sysobjects name in the tempdb adds 17 bytes to make the table name unique. If the table name is more than 238 bytes, the temporary table name in sysobjects uses only the first 238 bytes, then adds 17 bytes to make it unique.

In versions of Adaptive Server earlier than 15.0, temporary table names in sysobjects were 30 bytes. If you used a table name with fewer than 13 bytes, the name was padded with underscores (_) to 13 bytes, then another 17 bytes of other characters to bring the name up to 30 bytes.

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, the following command will return rows from MYTABLE, MyTable, or mytable, or any combination of uppercase and lowercase letters in the name:

```
select * from MYTABLE
```

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 253 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')
```

Do not not use:

```
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')
```

Syntax that includes quotes

When the quoted_identifier option is set to on, you do not need to use double quotes around an identifier if the syntax of the statement requires that a quoted string contain an identifier. For example:

```
set quoted_identifier on
create table '1one' (c1 int)
```

However, object_id() requires a string, so you must include the table name in quotes to select the information:

```
select object_id('1one')
-----------------------
  896003192
```

You can include an embedded double quote in a quoted identifier by doubling the quote:

```
create table "embedded""quote" (c1 int)
```

However, there is no need to double the quote when the statement syntax requires the object name to be expressed as a string:

```
select object_id('embedded"quote')
```

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"
```

Do not use:

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. Example 2 is incorrect because the syntax style for the column name does not match the syntax style used for the table name.

Example 1

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

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

Example 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 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 the OE ligature 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. For more information, see “Using wildcard characters with datetime data” on page 383.

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

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

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 4-7 summarizes the wildcard characters:

Table 4-7: 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 underscore (_) 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"
```

When using bracketed identifiers to create objects, such as with create table [*table_name*] or create dstabase [*dbname*], you must include at least one valid character.

All trailing spaces within bracketed identifiers are removed from the object name. For example, you achieve the same results executing the following create table commands:

- create table [tab1<space><space>]
- create table [tab1]

- create table [tab1<space><space><space>]
- create table tab1

This rule applies to all objects you can create using bracketed identifiers.

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 4-8 shows examples of square brackets used as escape characters with like.

Table 4-8: 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 4-9 shows examples of escape clauses used with like.

Table 4-9: 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 5 Reserved Words

Keywords, also known as reserved words, are words that have special meanings. This chapter lists Transact-SQL and ANSI SQL keywords.

Topics covered are:

Transact-SQL reserved words

The words in Table 5-1 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 in *Reference Manual: Procedures*.

Table 5-1: List of Transact-SQL reserved words

	Words
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, count_big, create, current, cursor
D	database, dbcc, deallocate, declare, decrypt, default, delete, desc, deterministic, disk, distinct, drop, dummy, dump
E	else, encrypt, end, endtran, errlvl, errordata, errorexit, escape, except, exclusive, exec, execute, exists, exit, exp_row_size, external
F	fetch, fillfactor, for, foreign, from
G	goto, grant, group
H	having, holdlock

	Words
I	identity, identity_gap, identity_start, if, in, index, inout, insensitive, insert, install, intersect, into, is, isolation
J	jar, join
K	key, kill
L	level, like, lineno, load, lock
M	materialized, max, max_rows_per_page, min, mirror, mirrorexit, modify
N	national, new, noholdlock, nonclustered, nonscrollable, non_sensitive, not, null, nullif, numeric_truncation Note Although “new” is not a Transact-SQL reserved word, since it may become a reserved word in the future, Sybase recommends that you avoid using it (for example, to name a database object). “New” is a special case (see “Potential ANSI SQL reserved words” on page 387 for information on other reserved words) because it appears in the spt_values table, and because sp_checkreswords displays “New” as a reserved word.
O	of, off, offsets, on, once, online, only, open, option, or, order, out, output, over
P	partition, perm, permanent, plan, 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, scroll, scrollable, select, semi_sensitive, set, setuser, shared, shutdown, some, statistics, stringsize, stripe, sum, syb_identity, syb_restree, syb_terminate
T	table, temp, temporary, textsize, to, tracefile, 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
X	xmlextract, xmlparse, xmltest, xmlvalidate

ANSI SQL reserved words

Adaptive Server includes entry-level ANSI SQL features. Full ANSI SQL 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 ANSI SQL features in subsequent releases. In addition, subsequent releases may include keywords not included in this list.

The words in Table 5-2 are ANSI SQL keywords that are not reserved words in Transact-SQL.

Table 5-2: List of ANSI SQL reserved words

	Words
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, semi_sensitive, 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 ANSI SQL 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 ANSI SQL reserved words in the future.

Table 5-3: List of potential ANSI SQL reserved words

	Words
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 6 **SQLSTATE Codes and Messages**

This chapter describes Adaptive Server's SQLSTATE status codes and their associated messages.

Topics covered are:

SQLSTATE codes are required for entry level ANSI SQL 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 ANSI SQL. In some cases, multiple Adaptive Server error conditions are associated with a single SQLSTATE value.

Warnings

Adaptive Server currently detects the following SQLSTATE warning conditions, described in Table 6-1:

Table 6-1: 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.
Warning–string data, right truncation	01004	Occurs when character, unichar, or binary data is truncated to 255 bytes. The data may be: • The result of a select statement in which the client does not support the WIDE TABLES property. • Parameters to an RPC on remote Adaptive Servers or Open Servers that do not support the WIDE TABLES property.

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 6-2 through Table 6-8. 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 6-2: 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 6-3: 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 6-4: 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.
`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 6-5: 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.

Message	Value	Description
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.
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 6-6: 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.

Message	Value	Description
`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.
`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 6-7: 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 6-8: 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

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

N

O

P

Q

R

S

T

U