SYBASE SQL Server™ Quick Reference Guide

SYBASE SQL Server Release 10.0

Document ID: 30200-01-1000-01

Change Level: 3

Last Revised: June 17, 1994

Document ID: 30200-01-1000

This publication pertains to Release 10.0 of the SYBASE database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of the agreement.

Document-Back Guarantee

Sybase welcomes corrections and comments on its documents. If you mark typographical errors, formatting errors, errors of fact, or areas that need clarification in any Sybase user's manual and send copies of marked-up pages to us, we will send you a clean copy of the manual, absolutely free**.**

Send pages to the Publications Operations Department at the address below. Please include your Site ID number.

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

(510) 922-3500
Fax (510) 922-5340

Document Orders

Customers may purchase additional copies of any document or the right to make photocopies of documentation for their in-house use.

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

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

Upgrades are provided only at regularly scheduled software release dates.

Table of Contents

Conventions

The font and syntax conventions in this reference are as follows:

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

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

Parentheses are typed as part of the command.	`compute` *`row_aggregate`* `(`*`column_name`*`)`
Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean choosing one or more of the enclosed options is optional. Do not type the brackets.	`[anchovies]`

The vertical bar means you can select only one of the options shown.	`{die_on_your_feet | live_on_your_knees |`
	`live_on_your_feet}`

The comma means you can choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.	`[extra_cheese, avocados, sour_cream]`

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.

Table 1: Font and Syntax Conventions

Formatting

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.

Case

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

In syntax statements, keywords (commands) are in normal font and identifiers and user-supplied words are in italics. You can disregard case when you type keywords:

“SELECT” is the same as “Select” is the same as “select”.

Expressions

Several different types of expressions are used in SQL Server™ syntax statements.

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

Table 2: Types of Expressions Used in Syntax Statements

Transact-SQL Commands

alter database

Increases the amount of space allocated to a database.

```
alter database database_name
   [on {default | database_device } [= size]
       [, database_device [= size]]...]
   [log on { default | database_device } [ = size ]
       [ , database_device [= size]]...]
   [with override]
   [for load]
```

alter table

Adds new columns and constraints, changes constraints, or drops constraints on an existing table.

```
alter table [database.[owner].]table_name
   {add column_name datatype
       [default {constant_expression | user | null}]
       {[{identity | null}]
       | [[constraint constraint_name]
           {{unique | primary key}
               [clustered | nonclustered]
               [with fillfactor = x] [on segment_name]
           | references [[database.]owner.]ref_table
               [(ref_column)]
           | check (search_condition)}]}...
       {[, next_column]}...

   | add {[constraint constraint_name]
       {unique | primary key}
           [clustered | nonclustered]
           (column_name [{, column_name}...])
           [with fillfactor = x] [on segment_name]
       | foreign key (column_name [{, column_name}...])
           references [[database.]owner.]ref_table
               [(ref_column [{, ref_column}...])]
       | check (search_condition)}

   | drop constraint constraint_name

   | replace column_name
       default {constant_expression | user | null}}
```

begin...end

Encloses a series of SQL statements so that control-of-flow language, such as **if ... else**, can affect the performance of the whole group.

```
begin
       statement block
   end
```

begin transaction

Marks the starting point of a user-defined transaction.

```
begin {transaction | tran } [transaction_name]
```

break

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

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

checkpoint

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

```
checkpoint
```

close

Deactivates a cursor.

```
close cursor_name
```

commit

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

compute Clause

Generates summary values that appear as additional rows in the query results. This allows you to see the detail and summary rows in one set of results. You can calculate summary values for subgroups, and you can calculate more than one aggregate for the same group.

Start of **select** statement
```
   compute row_aggregate (column_name)
       [, row_aggregate(column_name)]...
   [by column_name [, column_name]...]
```

continue

Causes the **while** loop to restart. **continue** is often activated by an **if** test.

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

create database

Creates a new database. Use **create database** from the *master* database.

```
create database database_name
   [on {default | database_device} [= size]
        [, database_device [= size]]...]
   [log on database_device [= size]
        [, database_device [= size]]...]
   [with override]
   [for load]
```

create default

Specifies a value to insert in a column (or in all columns of a user-defined datatype) if no value is explicitly supplied at insert time.

```
create default [owner.]default_name
   as constant_expression
```

create index

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

```
create [unique] [clustered | nonclustered]
        index index_name
   on [[database.]owner.]table_name (column_name
        [, column_name]...)
   [with {fillfactor = x, ignore_dup_key, sorted_data,
        [ignore_dup_row | allow_dup_row]}]
   [on segment_name]
```

create procedure

Creates a stored procedure that can take one or more user-supplied parameters.

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

create rule

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

```
create rule [owner.]rule_name
   as condition_expression
```

create schema

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

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

create table

Creates new tables and optional integrity constraints.

```
 create table [database.[owner].]table_name
   (column_name datatype
       [default {constant_expression | user | null}]
       {[{identity | null | not null}]
       | [[constraint constraint_name]
           {{unique | primary key}
               [clustered | nonclustered]
               [with fillfactor = x] [on segment_name]
           | references [[database.]owner.]ref_table
               [(ref_column)]
           | check (search_condition)}]}...

   | [constraint constraint_name]
       {{unique | primary key}
```

```
            [clustered | nonclustered]
            (column_name [{, column_name}...])
            [with fillfactor = x] [on segment_name]
        | foreign key (column_name [{, column_name}...])
            references [[database.]owner.]ref_table
                [(ref_column [{, ref_column}...])]
        | check (search_condition)}

    [{, {next_column | next_constraint}}...])

    [on segment_name]
```

create trigger

Creates a trigger, a type of stored procedure often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   {for {insert , update , delete}
   as SQL_statements
```

Or, using the **if update** clause:

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   for {insert , update}
   as
       [if update (column_name)
           [{and | or} update (column_name)]...]
           SQL_statements
       [if update (column_name)
           [{and | or} update (column_name)]...
           SQL_statements]...
```

create view

Creates a view, which is an alternative way of looking at the data in one or more tables.

```
create view [owner.]view_name
   [(column_name [, column_name]...)]
   as select [distinct] select_statement
   [with check option]
```

dbcc

Database Consistency Checker (**dbcc**) checks the logical and physical consistency of a database. **dbcc** should be used regularly as a periodic check, or if damage is suspected.

```
dbcc
   {checktable({table_name|table_id}[, skip_ncindex]) |
    checkdb [(database_name [, skip_ncindex] )] |

    checkalloc [(database_name [, fix | nofix])] |
    tablealloc ({table_name | table_id}
       [, {full | optimized | fast | null}
       [, fix | nofix ] ]] )|
    indexalloc ({table_name | table_id}, index_id
       [, {full | optimized | fast | null}
       [, fix | nofix ]] ) |

    checkcatalog [(database_name)] |

    dbrepair (database_name, dropdb) |

    reindex ({table_name | table_id}) |
    fix_text ({table_name | table_id}) }
```

deallocate cursor

Makes a cursor inaccessible and releases all memory resources committed to that cursor.

```
deallocate cursor cursor_name
```

declare

Declares the name and type of local variables for a batch or procedure. Local variables are assigned values with a **select** statement.

Variable declaration:

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

Variable assignment:

```
select @variable = {expression | select_statement}
   [, @variable = {expression | select_statement} ...]
   [from table_list]
   [where search_conditions]
   [group by group_by_list]
   [having search_conditions]
   [order by order_by_list]
   [compute function_list [by by_list]]
```

declare cursor

Defines a cursor.

```
declare cursor_name cursor
   for select_statement
   [for {read only | update [of column_name_list]}]
```

delete

Removes rows from a table.

```
delete [from] [[database.]owner.]{table_name|view_name}
   [where search_conditions]
```

```
delete [[database.]owner.]{table_name | view_name}
   [from [[database.]owner.]{table_name | view_name}
    [, [[database.]owner.]{table_name | view_name}]...]
   [where search_conditions]
```

```
delete [from] [[database.]owner.]{table_name|view_name}
   where current of cursor_name
```

disk init

Makes a physical device or file usable by SQL Server. (The master device is initialized by the **sybinit** installation program; it is not necessary to initialize this device with **disk init**.)

```
disk init
   name = "device_name" ,
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number ]
   [, contiguous] (OpenVMS only)
```

disk mirror

Creates a software mirror that immediately takes over when the primary device fails. You can mirror the master device, devices that store data, and devices that store transaction logs; you cannot mirror dump devices.

```
disk mirror
   name = "device_name" ,
   mirror = "physicalname"
   [ ,writes = { serial | noserial }]
   [ ,contiguous ] (OpenVMS only)
```

disk refit

Rebuilds the *master* database's *sysusages* and *sysdatabases* system tables from information contained in *sysdevices*. Use **disk refit** after **disk reinit** as part of the procedure to restore the master database.

```
disk refit
```

disk reinit

Rebuilds the *master* database's *sysdevices* system table. Use **disk reinit** as part of the procedure to restore the master database.

```
disk reinit
   name = "device_name",
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number]
```

disk remirror

Restarts disk mirroring after it is stopped by failure of a mirrored device or temporarily disabled by the **disk unmirror** command.

```
disk remirror
   name = "device_name"
```

disk unmirror

De-activates disk mirroring to allow hardware maintenance or the changing of a hardware device. **disk unmirror** disables either the original database device or the mirror, so that it is no longer available to SQL Server for reads or writes. It does not remove the associated file from the operating system.

```
disk unmirror
   name = "device_name"
   [ ,side = { “primary” | secondary }]
   [ ,mode = { retain | remove }]
```

drop database

Removes one or more databases from SQL Server.

```
drop database database_name [, database_name]...
```

drop default

Removes a user-defined default.

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

drop index

Removes an index from a table in the current database.

```
drop index table_name.index_name
   [, table_name.index_name]...
```

drop procedure

Removes user-defined stored procedures.

```
drop procedure [owner.]procedure_name
   [, [owner.]procedure_name] ...
```

drop rule

Removes a user-defined rule.

```
drop rule [owner.]rule_name [, [owner.]rule_name]...
```

drop table

Removes a table definition and all of its data, indexes, triggers, and permission specifications from the database.

```
drop table [[database.]owner.]table_name
   [, [[database.]owner.]table_name ]...
```

drop trigger

Removes a trigger.

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

drop view

Removes one or more views from the current database.

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

dump database

Makes a backup copy of the entire database, including the transaction log, in a form that can be read in with **load database**. Dumps and loads are performed through a Backup Server™.

```
dump database database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
        density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       notify = {client | operator_console}
       }]]
```

dump transaction

Makes a copy of a transaction log and removes the inactive portion.

To make a routine log dump:

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       notify = {client | operator_console}}]
```

To truncate the log without making a backup copy:

```
dump tran[saction] database_name
   with truncate_only
```

To truncate a log that is filled to capacity. **Use only as a last resort**:

```
dump tran[saction] database_name
   with no_log
```

To back up the log after a database device fails:

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
```

```
[with {
    density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    retaindays = number_days,
    [noinit | init],
    no_truncate,
    notify = {client | operator_console}}]
```

execute

Runs a system procedure or a user-defined stored procedure.

```
[execute] [@return_status = ]
   [[[server.]database.]owner.]procedure_name[;number]
       [[@parameter_name =] value |
           [@parameter_name =] @variable [output]
       [,[@parameter_name =] value |
           [@parameter_name =] @variable [output]...]]
   [with recompile]
```

fetch

Returns a row or a set of rows from a cursor result set.

```
fetch cursor_name [ into fetch_target_list ]
```

goto Label

Branches to a user-defined label.

```
label:
   goto label
```

grant

Assigns permissions to users.

To grant permission to access database objects:

```
grant {all [privileges]| permission_list}
   on { table_name [(column_list)]
       | view_name[(column_list)]
       | stored_procedure_name}
   to {public | name_list | role_name}
   [with grant option]
```

To grant permission to create database objects:

```
grant {all [privileges] | command_list}
   to {public | name_list | role_name}
```

group by and having Clauses

Used in **select** statements to divide a table into groups.

Start of **select** statement

```
group by [all] aggregate_free_expression
       [, aggregate_free_expression]...
   [having search_conditions]
```

End of **select** statement

if...else

Imposes conditions on the execution of a SQL statement. The statement following an **if** keyword and its condition is executed if the condition is satisfied (when the logical expression returns “true”). The optional **else** keyword introduces an alternate SQL statement that executes when the **if** condition is not satisfied (when the logical expression returns “false”).

```
if logical_expression
   statements
[else
   [if logical_expression]
   statement]
```

insert

Adds new rows to a table or view.

```
insert [into] [database.[owner.]]{table_name|view_name}
   [(column_list)]
   {values (expression [, expression]...)
       |select_statement }
```

kill

Kills a process.

```
kill spid
```

load database

Loads a backup copy of a user database, including its transaction log, that was created with **dump database.** The **listonly** and **headeronly** options display information about the dump files without loading them. Dumps and loads are performed through a Backup Server.

```
load database database_name
   from stripe_device [at backup_server_name ]
        [density = density_value,
         blocksize = number_bytes,
         dumpvolume = volume_name,
         file = file_name]
   [stripe on stripe_device [at backup_server_name ]
        [density = density_value,
         blocksize = number_bytes,
         dumpvolume = volume_name,
         file = file_name]
   [stripe on stripe_device [at backup_server_name ]
        [density = density_value,
         blocksize = number_bytes,
         dumpvolume = volume_name,
         file = file_name]...]
   [with {
        density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name,
        [dismount | nodismount],
        [nounload | unload],
        listonly [= full],
        headeronly,
        notify = {client | operator_console}
        }]
```

load transaction

Loads a backup copy of the transaction log that was created with the **dump transaction** command. The **listonly** and **headeronly** options display information about the dump files without loading them. Dumps and loads are performed through a Backup Server.

```
load tran[saction] database_name
   from stripe_device [at backup_server_name]
        [density = density_value,
         blocksize = number_bytes,
         dumpvolume = volume_name,
         file = file_name]
   [stripe on stripe_device [at backup_server_name]
        [density = density_value,
         blocksize = number_bytes,
         dumpvolume = volume_name,
         file = file_name]
```

```
[stripe on stripe_device [at backup_server_name]
    [density = density_value,
     blocksize = number_bytes,
     dumpvolume = volume_name,
     file = file_name]...]
[with {
    density = density_value,
    blocksize = number_bytes,
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    listonly [= full],
    headeronly,
    notify = {client | operator_console}
}]
```

open

Opens a cursor for processing.

```
open cursor_name
```

order by Clause

Returns query results in the specified column(s) in sorted order.

```
[order by {[table_name.| view_name.]column_name
      | select_list_number | expression} [asc | desc]
   [,{[table_name.| view_name.] column_name
       select_list_number|expression} [asc |desc]]...]
```

prepare transaction

Used by DB-Library™ in a two-phase commit application to see if a server is prepared to commit a transaction.

```
prepare transaction
```

print

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

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

raiserror

Prints a user-defined error message on the user's screen and sets a system flag to record that an error condition has occurred.

```
raiserror error_number
   [{format_string | @local_variable}] [, arg_list]
   [extended_value = extended_value [{,
   extended_value = extended_value}...]]
```

readtext

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

```
readtext [[database.]owner.]table_name.column_name
   text_pointer offset size [holdlock]
   [using {bytes | chars | characters}]
```

reconfigure

Sets configuration variables that control various aspects of SQL Server's memory allocation, performance, and options. Used with the system procedure **sp_configure**.

```
reconfigure [with override]
```

return

Exits from a batch or procedure unconditionally, optionally providing a return status. Statements following **return** are not executed.

```
return [integer_expression]
```

revoke

Revokes permissions from users.

```
revoke [grant option for]
   {all [privileges] | permission_list}
   on { table_name [(column_list)]
       | view_name [(column_list)]
       | stored_procedure_name}
   from {public | name_list | role_name}
   [cascade]

revoke {all [privileges] | command_list}
   from {public | name_list | role_name}
```

rollback

Rolls a user-defined transaction back to the last savepoint inside the transaction or to the beginning of the transaction.

```
rollback {transaction | tran | work}
   [transaction_name | savepoint_name]
```

rollback trigger

Rolls back the work done in a trigger, including the data modification that caused the trigger to fire, and issues an optional **raiserror** statement.

```
rollback trigger
   [with raiserror_statement]
```

save transaction

Sets a savepoint within a transaction.

```
save transaction savepoint_name
```

select

Retrieves rows from database objects.

```
select [all | distinct] select_list
   [into [[database.]owner.]table_name]
   [from [[database.]owner.]{table_name |view_name}
            [holdlock | noholdlock] [shared]
        [,[[database.]owner.]{table_name |view_name}
            [holdlock | noholdlock] [shared]]... ]

   [where search_conditions]

   [group by [all] aggregate_free_expression
        [, aggregate_free_expression]... ]
   [having search_conditions]

   [order by
   {[[[database.]owner.]{table_name.|view_name.}]
        column_name | select_list_number | expression}
            [asc | desc]
   [,{[[[database.]owner.]{table_name|view_name.}]
```

```
        column_name | select_list_number | expression}
            [asc | desc]]...]

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

    [for {read only | update [of column_name_list]}]

    [for browse]
```

set

Sets SQL Server query-processing options for the duration of the user's work session. Can be used to set some options inside a trigger or stored procedure.

```
set ansinull {on | off}

set ansi_permissions {on | off}

set arithabort [arith_overflow | numeric_truncation]
   {on | off}

set arithignore [arith_overflow] {on | off}

set {chained, close on endtran, nocount, noexec,
   parseonly, procid, self_recursion, showplan}
   {on | off}

set char_convert {off | on [with {error | no_error}] |
   charset [with {error | no_error}]}

set cursor rows number for cursor_name

set {datefirst number, dateformat format,
   language language}

set dup_in_subquery {on | off}

set fipsflagger {on | off}

set flushmessage {on | off}

set identity_insert [database.[owner.]]table_name
   {on | off}

set offsets {select, from, order, compute, table,
   procedure, statement, param, execute} {on | off}

set quoted_identifier {on | off}

set role {"sa_role" | "sso_role" | "oper_role"}
   {on | off}

set {rowcount number, textsize number}
set statistics {io, time} {on | off}
set string_rtruncation {on | off}
set textsize {number}
set transaction isolation level {1 | 3}
```

setuser

Allows a Database Owner to impersonate another user.

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

shutdown

Shuts down the SQL Server from which the command is issued, its local Backup Server, or a remote Backup Server. This command can only be issued by a System Administrator.

```
shutdown [srvname] [with {wait | nowait}]
```

truncate table

Removes all rows from a table.

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

union Operator

Returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the **all** keyword is specified.

```
     select select_list [into clause]
         [from clause] [where clause]
         [group by clause] [having clause]
 [union [all]
     select select_list
         [from clause] [where clause]
         [group by clause] [having clause] ]...
 [order by clause]
 [compute clause]
```

update

Changes data in existing rows, either by adding data or by modifying existing data.

```
update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
           {expression2|NULL|(select_statement)}]...
   [from [[database.]owner.]{table_name | view_name}
       [,[[database.]owner.]{table_name|view_name}]...]
   [where search_conditions]
```

```
update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
           {expression2|NULL|(select_statement)}]...
   where current of cursor_name
```

update statistics

Updates information about the distribution of key values in specified indexes.

```
update statistics table_name [index_name]
```

use

Specifies the database with which you want to work.

```
use database_name
```

waitfor

Specifies a specific time, a time interval, or an event for the execution of a statement block, stored procedure, or transaction.

```
waitfor { delay time | time time | errorexit
   | processexit | mirrorexit }
```

where Clause

Sets the search conditions in a **select**, **insert**, **update**, or **delete** statement. (Joins and subqueries are specified in the search conditions: see the “Joins” and “Subqueries” sections for full details.)

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

where [not] expression is [not] null

where [not]
   expression [not] between expression and expression

where [not]
   expression [not] in ({value_list | subquery})

where [not] exists (subquery)

where [not]
   expression comparison_operator {any|all} (subquery)

where [not] column_name join_operator column_name

where [not] boolean_expression

where [not] expression {and | or} [not] expression
```

while

Sets a condition for the repeated execution of a statement or statement block. The statement(s) are executed repeatedly as long as the specified condition is true.

```
while logical_expression
      statement
```

writetext

Permits non-logged, interactive updating of an existing text or image column.

```
writetext [[database.]owner.]table_name.column_name
   text_pointer [with log] data
```

Transact-SQL Functions

Aggregate Functions

The aggregate functions generate summary values that appear as new columns in the query results. They can be used in the select list or the **having** clause of a **select** statement or subquery, and often appear in a statement that includes a **group by** clause.

Aggregate Function	Result
sum([all | distinct] *expression*)	Total of (distinct) values in the numeric column
avg([all | distinct] *expression*)	Average of (distinct) values in the numeric column
count([all | distinct] *expression*)	Number of (distinct) non-null values in the column
count(*)	Number of selected rows
max(*expression*)	Highest value in the expression
min(*expression*)	Lowest value in the expression

Datatype Conversion Functions

Datatype conversion functions change expressions from one datatype to another and specify new display formats for date / time information. SQL Server provides three datatype conversion functions, **convert()**, **inttohex()**, and **hextoint()**, which can be used in the select list, in the **where** clause, and anywhere else an expression is allowed.

Function	Argument	Result
convert	(*datatype* [(*length*) | (*precision*[, *scale*])], *expression*[, *style*])	Converts between a wide variety of datatypes and reformats date / time and money data for display purposes.
hextoint	(*hexadecimal_string*)	Returns the platform-independent integer equivalent of a hexadecimal string.
inttohex	(*integer_expression*)	Returns the platform-independent hexadecimal equivalent of an integer.

Date Functions

Manipulate *datetime* values.

Function	Argument	Result
getdate	()	Returns the current system date and time.
datename	(*datepart*, *date*)	Returns the name of the specified part (such as the month "June") of a *datetime* value, as a character string. If the result is numeric, such as "23" for the day, it is still returned as a character string.
datepart	(*datepart*, *date*)	Returns an integer value for the specified part of a *datetime* value.
datediff	(*datepart*, *date1*, *date2*)	Returns *date2* - *date1*, measured in the specified date part.
dateadd	(*datepart*, *numeric_expression*, *date*)	Returns the date produced by adding the specified number of the specified date parts to the date. *numeric_expression* can be any numeric type; the value is truncated to an integer.

Mathematical Functions

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

Function	Argument	Result
abs	(*numeric*)	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.
acos	(*approx_numeric*)	Returns the angle (in radians) whose cosine is the specified value.
asin	(*approx_numeric*)	Returns the angle (in radians) whose sine is the specified value.

Function	Argument	Result
atan	(*approx_numeric*)	Returns the angle (in radians) whose tangent is the specified value**.**
atn2	(*approx_numeric1*, *approx_numeric2*)	Returns the angle (in radians) whose tangent is (*approx_numeric1/approx_numeric2).*
ceiling	(*numeric*)	Returns the smallest integer greater 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.
cos	(*approx_numeric*)	Returns the cosine of the specified angle (in radians).
cot	(*approx_numeric*)	Returns the cotangent of the specified angle (in radians).
degrees	(*numeric*)	Converts radians to degrees. Results are of the same type as the numeric expression. For *numeric* and *decimal* expressions, the results have an internal precision of 77 and a scale equal to that of the expression. When *money* datatypes are used, internal conversion to *float* may cause loss of precision.
exp	(*approx_numeric*)	Returns the exponential value of the specified value.
floor	(*numeric*)	Returns the largest integer 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.
log	(*approx_numeric*)	Returns the natural logarithm of the specified value.
log10	(*approx_numeric*)	Returns the base 10 logarithm of the specified value.
pi	()	Returns the constant value of 3.1415926535897936.
power	(*numeric*, *power*)	Returns the value of *numeric* raised to the power *power.* Results are of the same type as *numeric*. For expressions of type *numeric* or *decimal*, the results have an internal precision of 77 and a scale equal to that of the expression.

Function	Argument	Result
radians	(*numeric*)	Converts degrees to radians. Results are of the same type as *numeric*. For expressions of type *numeric* or *decimal*, the results have an internal precision of 77 and a scale equal to that of the numeric expression. When money datatypes are used, internal conversion to *float* may cause loss of precision.
rand	([*integer*])	Returns a random *float* value between 0 and 1, using the optional *integer* as a seed value.
round	(*numeric*, *integer)*	Rounds the *numeric* so that it has *integer* significant digits. A positive integer determines the number of significant digits to the right of the decimal point; a negative *integer*, the number of significant digits to the left of the decimal point. Results are of the same type as the numeric expression and, for *numeric* and *decimal* expressions, have an internal precision of 77 and scale equal to that of the numeric expression.
sign	(*numeric*)	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.
sin	(*approx_numeric*)	Returns the sine of the specified angle (measured in radians).
sqrt	(*approx_numeric*)	Returns the square root of the specified value.
tan	(*approx_numeric*)	Returns the tangent of the specified angle (measured in radians).

Row Aggregate Functions

Generate summary values that appear as additional rows in the query results.

Start of select statement

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

Name	Meaning
sum	Total of values in the (numeric) column
avg	Average of values in the (numeric) column
min	Lowest value in the column
max	Highest value in the column
count	Number of non-null values in the column

String Functions

Operate on binary data, character strings, and expressions. String functions can be nested, and they can be used 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.

Function	Argument	Result
ascii	(*char_expr*)	Returns the ASCII code for the first character in the expression.
char	(*integer_expr)*	Converts a single-byte *integer* value to a *character* value. (**char** is usually used as the inverse of **ascii**.) *integer_expr* must be between 0 and 255. Returns a *char* datatype. If the resulting value is the first byte of a multibyte character, the character may be undefined.
charindex	(*expression1, expression2)*	Searches *expression2* for the first occurrence of *expression1* and returns an integer representing its starting position. If *expression1* is not found, returns 0. If *expression1* contains wildcard characters, **charindex** treats them as literals.
char_length	(*char_expr)*	Returns an integer representing the number of characters in a character expression or *text* value. For variable-length data, **char_length** strips the expression of trailing blanks before counting the number of characters. For multi-byte character sets, the number of characters in the expression is usually less than the number of bytes; use **datalength** (See "System Functions") to determine the number of bytes.

Function	Argument	Result
difference	(*char_expr1, char_expr2)*	Returns an integer representing the difference between two **soundex** values. See **soundex**, below.
lower	(*char_expr)*	Converts uppercase letters to lowercase, returning a character value.
ltrim	(*char_expr)*	Removes leading blanks from the character expression. Only values equivalent to the space character in the current character set are removed.
patindex	("%*pattern*%", *char_expr* [, **using** {**bytes | chars | characters**}])	Returns an integer representing the starting position of the first occurrence of *pattern* in the specified character expression, or a zero if *pattern* is not found. By default, **patindex** returns the offset in characters; to return the offset in bytes (multibyte character strings), specify **using bytes**. The % wildcard character must precede and follow *pattern* (except when searching for first or last characters). See "Wildcard Characters" for a description of the wildcard characters that can be used in *pattern*. Can be used on *text* data.
replicate	(*char_expr, integer_expr)*	Returns a string with the same datatype as *char_expr*, containing the same expression repeated the specified number of times or as many times as will fit into a 255 byte space, whichever is less.
reverse	(*char_expr)*	Returns the reverse of *char_expr*; if *char_expr* is "abcd", it returns "dcba".
right	(*char_expr, integer_expr)*	Returns the part of the character expression starting the specified number of *characters* from the right. Return value has the same datatype as the character expression.
rtrim	(*char_expr)*	Removes trailing blanks. Only values equivalent to the space character in the current character set are removed.
soundex	(*char_expr)*	Returns a four-character **soundex** code for character strings that are composed of a contiguous sequence of valid single- or double-byte roman letters.
space	(*integer_expr)*	Returns a string with the indicated number of single-byte spaces.

Function	Argument	Result
str	(*approx_numeric* [, *length* [, *decimal*]])	Returns a character representation of the floating point number. *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); *decimal* sets the number of decimal digits to be returned. *length* and *decimal* are optional. If given, they must be non-negative. Default *length* is 10; default *decimal* is 0. **str()** rounds the decimal portion of the number so that the results fit within the specified *length*.
stuff	(*char_expr1, start, length, char_expr2*)	Deletes *length* characters from *char_expr1* at *start*, then inserts *char_expr2* into *char_expr1* at *start*. To delete characters without inserting other characters, *char_expr2* should be NULL (not "", which indicates a single space).
substring	(*expression, start, length*)	Returns part of a character or binary string. *start* specifies the character position at which the substring begins. *length* specifies the number of characters in the substring.
upper	(*char_expr*)	Converts lower to upper case, returning a character value.

System Functions

Return special information from the database.

Function	Argument	Result
col_name	(*object_id, column_id* [, *database_id*])	Returns the column name.
col_length	(*object_name, column_name*)	Returns the defined length of column. Use **datalength** to see the actual data size.

Function	Argument	Result
curunreservedpgs	(*database_id*, *page_number*, *free_pages*)	Returns the number of free pages in the disk piece that contains *page_number*. Use *free_pages* to specify a default value to be returned by the function. If the database is open, **curunreservedpgs** replaces this value with the actual number of free pages stored in memory for that disk piece.
data_pgs	(*object_id*, {*doampg* | *ioampg*})	Returns the number of pages used by table (*doampg)* or index (*ioampg*). The result does not include pages used for internal structures.
datalength	(*expression*)	Returns the length of *expression* in bytes. *expression* is usually a column name. If *expression* is a character constant, it must be enclosed in quotes.
db_id	([*database_name*])	Returns the database ID number. *database_name* must be a character expression; if it is a constant expression, it must be enclosed in quotes. If no *database_name* is supplied, **db_id** returns the ID number of the current database.
db_name	([*database_id*])	Returns the database name. *database_id* must be a numeric expression. If no *database_id* is supplied, **db_name** returns the name of the current database.
host_id	( )	Returns the host process ID of the client process (not the Server process).
host_name	( )	Returns the current host computer name of the client process (not the Server process).
index_col	(*object_name*, *index_id*, *key_#* [, *user_id*])	Returns the name of the indexed column; returns NULL if *object_name* is not a table or view name.

Function	Argument	Result
isnull	(*expression1*, *expression2*)	Substitutes the value specified in *expression2* when *expression1* evaluates to NULL.The datatypes of the expressions must convert implicitly, or you must use the **convert** function.
lct_admin	({{ "**lastchance**" | "**logfull**" | "**unsuspend**"} , *database_id*} | "**reserve**", *log_pages*})	Manages the log segment's last-chance threshold. **lastchance** creates a last-chance threshold in the specified database. **logfull** returns 1 if the last-chance threshold has been crossed in the specified database and 0 if it has not. **unsuspend** awakens suspended tasks in the database and disables the last-chance threshold if that threshold has been crossed. **reserve** returns the number of free log pages required to successfully dump a transaction log of the specified size.
object_id	(*object_name*)	Returns the object ID.
object_name	(*object_id*[, *database_id*])	Returns the object name.
proc_role	("**sa_role**" | "**sso_role**" | "**oper_role**")	Checks to see if the invoking user possesses the correct role to execute the procedure. Returns 1 if the invoker has the required role. Otherwise, returns 0.
reserved_pgs	(*object_id*, {*doampg* | *ioampg*})	Returns the number of pages allocated to table or index. This function **does** report pages used for internal structures.
rowcnt	(*doampg*)	Returns the number of rows in a table (estimate).
show_role	( )	Returns the user's current active roles, if any (**sa_role**, **sso_role**, or **oper_role**). If the user has no roles, returns NULL.

Function	Argument	Result
suser_id	([*server_user_name*])	Returns the server user's ID number from *syslogins*. If no *server_user_name* is supplied, it returns the server ID of the current user.
suser_name	([*server_user_id*])	Returns the server user's name. Server user's ID's are stored in *syslogins*. If no *server_user_id* is supplied, it returns the name of the current user.
used_pgs	(*object_id*, *doampg*, *ioampg*)	Returns the total number of pages used by a table and its clustered index.
tsequal	(*timestamp*, *timestamp2*)	Compares *timestamp* values to prevent update on a row that has been modified since it was selected for browsing. *timestamp* is the timestamp of the browsed row; *timestamp2* is the timestamp of the stored row. Allows you to use browse mode without calling DB-Library. (See "Browse Mode.")
user		Returns the user's name.
user_id	([*user_name*])	Returns the user's ID number. Reports the number from *sysusers* in the current database. If no *user_name* is supplied, it returns the ID of the current user.
user_name	([*user_id*])	Returns the user's name, based on the user's ID in the current database. If no *user_id* is supplied, it returns the name of the current user.
valid_name	(*character_expression*)	Returns 0 if *string* is not a valid identifier (illegal characters or *string* is more than 30 bytes long), a nonzero number if *string* is a valid identifier.

text and *image* Functions

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

Function	Argument	Result
patindex	("*%pattern%*", *char_expr* [, **using** {**bytes** | **chars** | **characters**}])	Returns an integer value representing the starting position of the first occurrence of *pattern* in the specified character expression, or zero if *pattern* is not found. By default, **patindex** returns the offset in characters; to return the offset in bytes for multibyte character strings, specify **using bytes**. The % wildcard character must precede and follow *pattern*, except when searching for first or last characters. See "Wildcard Characters" for a description of the wildcard characters that can be used in *pattern*.
textptr	(*text_columnname*)	Returns the text pointer value, a 16-byte binary value. The text pointer is checked to ensure that it points to the first text page.
textvalid	("*table_name.col_name*", *textpointer*)	Checks that a given text pointer is valid. Note that the identifier for a *text* or *image* column must include the table name. Returns 1 if the pointer is valid, or 0 if the pointer is invalid.

Transact-SQL Topics

Comments

Attach explanatory text to SQL statements, statement blocks, and system procedures. Comments are not executed.

A comment can be inserted on a line by itself or at the end of a command line. Two comment styles are available: the "slash-asterisk" style:

```
/* text of comment (slash-asterisk style) */
```

and the "double-hyphen" style:

```
-- text of comment (double-hyphen style)
```

Cursors

A cursor provides access to the set of rows returned by a SQL query. A cursor is a symbolic name that is associated with a **select** statement. Cursors enable you to access individual rows of data returned by SQL Server. Cursors consist of two parts: the **cursor result set** and the **cursor position**.

To create a cursor, use the **declare cursor** statement:

```
declare cursor_name cursor
   for select_statement
   [for {read only | update [of column_name_list]}]
```

To open the cursor:

```
open cursor_name
```

After opening the cursor, you can fetch a row:

```
fetch cursor_name [into fetch_target_list]
```

When you are finished with the result set of a cursor, you can **close** it:

```
close cursor_name
```

If you want to discard the cursor, you must **deallocate** it:

```
deallocate cursor cursor_name
```

Datatypes

Specify the type of information, size, and storage format of columns, stored procedure parameters, and local variables.

Datatypes by Category	Synonyms	Range	Bytes of Storage
Exact numeric: integers			
tinyint		0 to 255	1
smallint		2^{15} -1 (32,767) to -2^{15} (-32,768)	2
int	*integer*	2^{31} (2,147,483,647) to -2^{31} (-2,147,483,648)	4
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 or 8
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

Datatypes by Category	Synonyms	Range	Bytes of Storage
Date/time			
smalldatetime		January 1, 1900 to June 6, 2079	4
datetime		January 1, 1753 to December 31, 9999	8
Character			
char(n)	*character*	255 characters or less	*n*
varchar(n)	*character varying, char varying*	255 characters or less	actual entry length
nchar(n)	*national character, national char*	255 characters or less	*n * @@ncharsize*
nvarchar(n)	*nchar varying, national char varying, national character varying*	255 characters or less	*n*
text		2^{31} -1 (2,147,483,647) bytes or less	0 or multiple of 2K (4K on Stratus)
Binary			
binary(n)		255 bytes or less	*n*
varbinary(n)		255 bytes or less	actual entry length
image		2^{31} -1 (2,147,483,647) bytes or less	0 or multiple of 2K (4K on Stratus)
Bit			
bit		0 or 1	1 (one byte holds up to 8 *bit* columns)

Disk Mirroring

Creates a software mirror of a user database device, the master database device, or a database device used for user database transaction logs. If a database device fails, its mirror immediately takes over.

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.

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*

SQL Server uses the following arithmetic operators:

Symbol	Meaning
+	addition
–	subtraction
*	multiplication
/	division
%	modulo (Transact-SQL extension)

SQL Server uses the following bitwise operators:

Symbol	Meaning
&	bitwise and (two operands)
|	bitwise or (two operands)
^	bitwise exclusive or (two operands)
~	bitwise not, unary not (one operand)

SQL Server uses the following comparison operators:

Symbol	Meaning
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to
!=	not equal to (Transact-SQL extension)
!>	not greater than (Transact-SQL extension)
!<	not less than (Transact-SQL extension)

Identifiers

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

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

IDENTITY Columns

IDENTITY columns contain system-generated values that uniquely identify each row within a table. They are used to store sequential numbers, such as invoice numbers or employee numbers, that are generated automatically by SQL Server. The value of the IDENTITY column uniquely identifies each row in a table.

Each table in a database can have a single IDENTITY column, of type numeric and scale zero. You can define an IDENTITY column when you create a table with a **create table** or **select into** statement, or add it later with an **alter table** statement.

By definition, IDENTITY columns cannot be updated and do not allow nulls. Each time you insert a row into a table, SQL Server automatically supplies a unique, sequential value for its IDENTITY column, beginning with the value "1". Manual insertions, deletions, transaction rollbacks, and server failures can create gaps in IDENTITY column values.

Joins

Joins compare two or more tables (or views) by specifying a column from each, comparing the values in those columns row by row, and concatenating rows that have matching values. Joins can also be stated as subqueries. (See "Subqueries" for more information.)

A join can be embedded in a **select**, **update**, **insert**, **delete**, or *subquery*. Other search conditions and clauses may follow the join condition(s). Joins use the following syntax:

```
Start of select, update, insert, delete, or subquery
   from {table_list | view_list}
   where [not]
       [table_name.| view_name.]column_name
           join_operator
       [table_name. | view_name.]column_name
   [{and | or} [not]
       [table_name.|view_name.]column_name
           join_operator
       [table_name.|view_name.]column_name]...
End of select, update, insert, delete, or subquery
```

Null Values

Marks columns having an unknown value (as opposed to those that have 0 or blank as a value). NULL allows you to distinguish between a deliberate entry of zero (for numerical columns) or blank (for character columns) and a non-entry (NULL for both numerical and character columns).

In **create table** statements, declare each columns as **null**, **not null**, or **identity** or accept the default null type for the database:

```
column_name datatype [null | not null | identity]
```

When adding a column to an existing table, you can specify it as **null** or **identity**:

```
column_name datatype [null | identity]
```

You can explicitly insert NULL into a column:

```
values({expression | null}
       [, {expression | null}]...)
```

Use the **update** statement to set a column value to NULL. Its syntax is:

```
set column_name = {expression | null}
       [, column_name = {expression | null}]...
```

In the **create procedure** statement, you can declare NULL as the default value for individual parameters:

```
create procedure procedure_name
       @param datatype [ = null ]
       [, @param datatype [ = null ]]...
```

Use the **isnull** built-in function to substitute a particular value for nulls. The substitution is made only for display purposes; actual column values are not affected. The syntax is:

```
isnull(expression, value)
```

Parameters

Parameters are arguments to a stored procedure. You define parameters when you create the procedure and supply their values when you execute the procedure. Not all procedures require parameters.

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

```
[execute] [@return_status = ]
   [[server.]database.]owner.]procedure_name[;number]
   [[@parameter_name =] value |
       [@parameter_name =] @variable [output]
   [,[@parameter_name =] value |
       [@parameter_name =] @variable [output]... ]]
   [with recompile]
```

Subqueries

Used to “nest” a **select** statement inside a **select**, **insert**, **update**, or **delete** statement, another subquery, or anywhere an expression is allowed (if it returns a single value). A subquery is always enclosed in parentheses.

```
(select [all | distinct] subquery_select_list
   [from [[database.]owner.]{table_name | view_name}
       [holdlock]
   [, [[database.]owner.]{table_name | view_name}
       [holdlock]]...]
   [where search_conditions]
   [group by aggregate_free_expression
       [, aggregate_free_expression]...]
   [having search_conditions])
```

text and image Datatypes

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

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

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

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

Variables (Local and Global)

Variables are defined entities that are assigned values. SQL Server has two kinds of variables:

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

```
select @variable_name = expression
      [ , @variable_name = expression ]...
   [from clause] [where clause] [group by clause]
   [having clause] [order by clause] [compute clause]
```

Wildcard Characters

Represent one or more characters, or a range of characters, in a match string.

Use wildcards 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"]
```

System Procedures

sp_addalias

Allows a SQL Server user to be known in a database as another user.

```
sp_addalias login_name, name_in_db
```

sp_addauditrecord

Allows users to enter user-defined audit records (comments) into the audit trail.

```
sp_addauditrecord [@text="message text"]
   [, @db_name="db_name"] [, @obj_name="object_name"]
   [, @owner_name="object_owner"]
   [, @dbid=database_ID] [, @objid=object_ID]
```

sp_addgroup

Adds a group to a database. Groups are used as collective names in granting and revoking privileges.

```
sp_addgroup grpname
```

sp_addlanguage

Defines the names of the months and days for an alternate language and its date format.

```
sp_addlanguage language, alias, months, shortmons,
   days, datefmt, datefirst
```

sp_addlogin

Adds a new user account to SQL Server.

```
sp_addlogin login_name, password [, defdb
    [, deflanguage [, fullname]]]
```

sp_addmessage

Adds user-defined messages to *sysusermessages* for use by stored procedure **print** and **raiserror** calls and by **sp_bindmsg**.

```
sp_addmessage message_num, message_text [, language]
```

sp_addremotelogin

Authorizes a new remote server user by adding an entry to *master.dbo.sysremotelogins*.

```
sp_addremotelogin remoteserver [, login_name
   [, remote_name] ]
```

sp_addsegment

Defines a segment on a database device in the current database.

```
sp_addsegment segname, dbname, devname
```

sp_addserver

Defines a remote server, or defines the name of the local server.

```
sp_addserver srvname [, {local | null}
   [, network_name]]
```

sp_addthreshold

Creates a threshold to monitor space on a database segment. When free space on the segment falls below the specified level, SQL Server executes the associated stored procedure.

```
sp_addthreshold database, segment, free_pages,
   procedure
```

sp_addtype

Creates a user-defined datatype.

```
sp_addtype typename,
   phystype [(length) | (precision [, scale])]
   [, "identity" | nulltype]
```

sp_addumpdevice

Adds a dump device to SQL Server.

```
sp_addumpdevice {"tape" | "disk"}, device_name,
   physicalname [, size]
```

sp_adduser

Adds a new user to the current database.

```
sp_adduser login_name [, name_in_db [, grpname]]
```

sp_auditdatabase

Establishes auditing of different types of events within a database, or of references to objects within that database from another database.

```
sp_auditdatabase [dbname [, "ok | fail | both | off"
   [, {"d u g r t o"}]]]
```

sp_auditlogin

Audits a SQL Server user’s attempts to access tables and views; audits the text of a user’s command batches; lists users on which auditing is enabled; gives the auditing status of a user; or displays the status of table, view, or command text auditing.

```
sp_auditlogin [login_name [, "table" | "view"
   [, "ok" | "fail" | "both" | "off"]]]
```

```
sp_auditlogin [login_name [, "cmdtext"
   [, "on" | "off"]]]
```

sp_auditobject

Audits accesses to tables and views.

```
sp_auditobject objname, dbname
   [, {"ok" | "fail" | "both" | "off"}
       [, "{d i s u}"]]
```

```
sp_auditobject {"default table"|"default view"},
   dbname [, {"ok" | "fail" | "both" | "off"}
   [, "{d i s u}"]]
```

sp_auditoption

Enables or disables system-wide auditing and global audit options, or reports on the status of audit options.

```
sp_auditoption {"all" | "enable auditing" | "logouts"
   | "server boots" | "adhoc records"}
   [, {"on" | "off"}]
```

```
sp_auditoption {"logins" | "rpc connections" |
   "roles"} [, {"ok" | "fail" | "both" | "off"}]
```

```
sp_auditoption "errors" [, {"nonfatal" | "fatal"
   | "both"}]
```

```
sp_auditoption "{sa | sso | oper | navigator |
      replication} commands"
   [, {"ok" | "fail" | "both" | "off"}]
```

sp_auditsproc

Audits the execution of stored procedures and triggers.

```
sp_auditsproc [sproc_name | "all", dbname
   [, {"ok" | "fail" | "both" | "off"}]]
```

```
sp_auditsproc "default", dbname
   [, {"ok" | "fail" | "both" | "off"}]
```

sp_bindefault

Binds a default to a column or user-defined datatype.

```
sp_bindefault defaultname, objectname [, futureonly]
```

sp_bindmsg

Binds a user message to a referential integrity constraint or check constraint.

```
sp_bindmsg constraint_name, message_num
```

sp_bindrule

Binds a rule to a column or user-defined datatype.

```
sp_bindrule rulename, objectname [, futureonly]
```

sp_changedbowner

Changes the owner of a database. **Do not** change the owner of the *sybsystemprocs* database.

```
sp_changedbowner login_name [, true ]
```

sp_changegroup

Changes a user's group.

```
sp_changegroup grpname, name_in_db
```

sp_checknames

Checks the current database for names that contain characters not in the 7-bit ASCII set.

```
sp_checknames
```

sp_checkreswords

Detects and displays identifiers that are Transact-SQL reserved words. Checks server names, device names, database names, segment names, user-defined datatypes, object names, column names, user names, login names, and remote login names.

```
sp_checkreswords [username]
```

sp_clearstats

Initiates a new accounting period for all server users or for a specified user. Prints statistics for the previous period by executing **sp_reportstats**.

```
sp_clearstats [user_name]
```

sp_commonkey

Defines a common key—columns that are frequently joined—between two tables or views.

```
sp_commonkey tabaname, tabbname, col1a, col1b
    [, col2a, col2b, ..., col8a, col8b]
```

sp_configure

Displays or changes configuration variables.

sp_configure [*config_name* [, *config_value*]]

sp_cursorinfo

Reports information about a specific cursor or all cursors that are active.

sp_cursorinfo [{*cursor_level* | null}] [*, cursor_name*]

sp_dboption

Displays or changes database options.

sp_dboption [*dbname*, *optname*, {true | false}]

sp_dbremap

Forces SQL Server to recognize changes made by **alter database**. Run this procedure only if instructed to do so by SQL Server messages.

sp_dbremap database_name

sp_depends

Displays information about database object dependencies—the view(s), trigger(s), and procedure(s) that depend on a specified table or view, and the table(s) and view(s) that the specified view, trigger, or procedure depends on.

sp_depends *objname*

sp_diskdefault

Sets a database device’s status to **defaulton** or **defaultoff**. This indicates whether or not a database device can be used for database storage if the user does not specify a database device or specifies **default** with the **create database** or **alter database** commands.

sp_diskdefault *logical_name* {defaulton | defaultoff}

sp_displaylogin

Displays information about a login account.

sp_displaylogin [*login_name*]

sp_dropalias

Removes the alias user name identity established with **sp_addalias**.

sp_dropalias *login_name*

sp_dropdevice

Drops a SQL Server database device or dump device.

sp_dropdevice *device_name*

sp_dropgroup

Drops a group from a database.

sp_dropgroup *grpname*

sp_dropkey

Removes from the *syskeys* table a key that had been defined using **sp_primarykey**, **sp_foreignkey**, or **sp_commonkey**.

sp_dropkey *keytype*, *tabaname* [, *tabbname]*

sp_droplanguage

Drops an alternate language from the server and removes its row from *master.dbo.syslanguages*.

sp_droplanguage *language* [, dropmessages]

sp_droplogin

Drops a SQL Server user login by deleting the user's entry in *master.dbo.syslogins*.

sp_droplogin *login_name*

sp_dropmessage

Drops user-defined messages from *sysusermessages*.

sp_dropmessage *message_number [, language*]

sp_dropremotelogin

Drops a remote user login.

```
sp_dropremotelogin remoteserver [, login_name
   [, remotename] ]
```

sp_dropsegment

Drops a segment from a database or unmaps a segment from a particular database device.

sp_dropsegment *segname*, *dbname* [, *devname*]

sp_dropserver

Drops a server from the list of known servers.

sp_dropserver ***server*** **[, droplogins]**

sp_dropthreshold

Removes a free-space threshold from a segment.

sp_dropthreshold ***database, segment, free_pages***

sp_droptype

Drops a user-defined datatype.

sp_droptype ***typename***

sp_dropuser

Drops a user from the current database.

sp_dropuser ***name_in_db***

sp_estspace

Estimates the amount of space required for a table and its indexes, and the time needed to create the index.

```
sp_estspace table_name, no_of_rows [, fill_factor
   [, cols_to_max [, textbin_len [, iosec]]]]
```

sp_extendsegment

Extends the range of a segment to another database device, or extends an existing segment on the current database device.

sp_extendsegment ***segname*****,** ***dbname, devname***

sp_foreignkey

Defines a foreign key on a table or view in the current database.

```
sp_foreignkey tabname, pktabname, col1 [, col2] ...
   [, col8]
```

sp_getmessage

Retrieves stored message strings from *sysmessages* and *sysusermessages* for **print** and **raiserror** statements.

sp_getmessage ***message_num*****,** ***@msg_var*** **output [,** ***language*****]**

sp_help

Reports information about a database object (any object listed in *sysobjects*), and about SQL Server-supplied or user-defined datatypes.

sp_help [*objname*]

sp_helpconstraint

Reports information about any integrity constraints specified for a table. This information includes the constraint name and the definition of the default, unique / primary key constraint, referential constraint, or check constraint.

sp_helpconstraint *tabname* [, detail]

sp_helpdb

Reports information about a particular database or about all databases.

sp_helpdb [*dbname*]

sp_helpdevice

Reports information about a particular device or about all SQL Server database devices and dump devices.

sp_helpdevice [*device_name*]

sp_helpgroup

Reports information about a particular group or about all groups in the current database.

sp_helpgroup [*grpname*]

sp_helpindex

Reports information about the indexes created on a table.

sp_helpindex *tabname*

sp_helpjoins

Lists the columns in two tables or views that are likely join candidates.

sp_helpjoins *lefttab*, *righttab*

sp_helpkey

Reports information about a primary, foreign, or common key of a particular table or view, or about all keys in the current database.

sp_helpkey [*objname*]

sp_helplanguage

Reports information about a particular alternate language or about all languages.

sp_helplanguage [*language*]

sp_helplog

Reports the name of the device that contains the first page of the log.

sp_helplog

sp_helpremotelogin

Reports information about a particular remote server's logins or about all remote servers' logins.

sp_helpremotelogin [*remoteserver* [,*remotename*]]

sp_helprotect

Reports on permissions for database objects, users, or groups.

sp_helprotect [*name* [, *name_in_db* [, "grant"]]]

sp_helpsegment

Reports information on a particular segment or on all of the segments in the current database.

sp_helpsegment [*segname*]

sp_helpserver

Reports information about a particular remote server or about all remote servers.

sp_helpserver [*server*]

sp_helpsort

Displays SQL Server's default sort order and character set.

sp_helpsort

sp_helptext

Prints the text of a system procedure, trigger, view, default, rule, or integrity check constraint.

sp_helptext *objname*

sp_helpthreshold

Reports the segment, free-space value, status, and stored procedure associated with all thresholds in the current database or all thresholds for a particular segment.

sp_helpthreshold [*segment_name*]

sp_helpuser

Reports information about a particular user or about all users in the current database.

sp_helpuser [*name_in_db*]

sp_indsuspect

Checks user tables for indexes that have been marked as suspect during recovery following a sort order change.

sp_indsuspect [*table_name*]

sp_lock

Reports information about processes that currently hold locks.

sp_lock [*spid1* [, *spid2*]]

sp_locklogin

Locks a SQL Server account so that the user cannot log in, or displays a list of all locked accounts.

sp_locklogin [*login_name*, "{lock | unlock}"]

sp_logdevice

Puts the system table *syslogs*, which contains the transaction log, on a separate database device.

sp_logdevice *dbname*, *device_name*

sp_modifylogin

Modifies the default database, default language, or full name for a SQL Server login account.

sp_modifylogin *login_name*, *option*, *value*

sp_modifythreshold

Modifies a threshold by associating it with a different threshold procedure, level of free space, or segment. You **cannot** use **sp_modifythreshold** to change the amount of free space or the segment name for the last-chance threshold.

sp_modifythreshold ***database, segment, free_pages***
   [*, new_procedure*] [, *new_free_pages*]
   [, *new_segment*]

sp_monitor

Displays statistics about SQL Server.

sp_monitor

sp_password

Adds or changes a password for a SQL Server login account.

sp_password ***caller_passwd*, *new_passwd*** **[, *login_name*]**

sp_placeobject

Puts future space allocations for a table or index on a particular segment.

sp_placeobject ***segname*, *objname***

sp_primarykey

Defines a primary key on a table or view.

sp_primarykey ***tabname*, *col1*** **[, *col2*, *col3*, ..., *col8*]**

sp_procxmode

Displays or changes the transaction modes associated with stored procedures.

sp_procxmode [*procedure_name* [, *transaction_mode*]]

sp_recompile

Causes each stored procedure and trigger that uses the named table to be recompiled the next time it runs.

sp_recompile ***tabname***

sp_remap

Remaps a Release 4.8 or later stored procedure, trigger, rule, default, or view to be compatible with Release 10.0. Use **sp_remap** on objects that the Release 10.0 upgrade procedure failed to remap.

sp_remap ***object_name***

sp_remoteoption

Displays or changes remote login options.

```
sp_remoteoption [remote_server, login_name,
   remote_name, opt_name, {true | false}]
```

sp_rename

Changes the name of a user-created object in the current database.

sp_rename ***objname*****,** ***newname***

sp_renamedb

Changes the name of a database. You **cannot** rename system databases or databases with external referential integrity constraints.

sp_renamedb ***dbname*****,** ***newname***

sp_reportstats

Reports statistics on system usage.

sp_reportstats [***user_name*****]**

sp_role

Grants or revokes roles to a SQL Server login account.

```
sp_role {"grant" | "revoke"},
   {sa_role | sso_role | oper_role}, login_name
```

sp_serveroption

Displays or changes remote server options.

sp_serveroption [***server*****,** ***optname*****, {true | false}]**

sp_setlangalias

Assigns or changes the alias for an alternate language.

sp_setlangalias ***language*****,** ***alias***

sp_spaceused

Displays the number of rows, the number of data pages, and the space used by one table or by all tables in the current database.

sp_spaceused [*tablename]*

sp_syntax

Displays the syntax of Transact-SQL statements, system procedures, utilities, and other routines (depending on which products and corresponding **sp_syntax** scripts exist on your server).

```
sp_syntax {command | fragment} [, module_name]
   [, language]
```

sp_thresholdaction

Executes automatically when the number of free pages on the log segment falls below the last-chance threshold (unless the threshold has been associated with a different procedure). Sybase does not provide this procedure.

```
sp_thresholdaction @dbname,
       @segment_name,
       @space_left,
       @status
```

sp_unbindefault

Unbinds a created default value from a column or from a user-defined datatype.

sp_unbindefault *objname* [, futureonly]

sp_unbindmsg

Unbinds a user-defined message from a constraint.

sp_unbindmsg *constraint_name*

sp_unbindrule

Unbinds a rule from a column or from a user-defined datatype.

sp_unbindrule *objname* [, futureonly]

sp_volchanged

Notifies the Backup Server that the operator performed the requested volume handling during a dump or load.

```
sp_volchanged session_id, device_name, action
   [, filename [, volume_name]]
```

sp_who

Reports information about all current SQL Server users and processes, or about a particular user or process.

sp_who [*login_name* | "*spid*"]

Catalog Stored Procedures

sp_column_privileges

Returns permissions information for one or more columns in a table or view.

sp_column_privileges *table_name* [, *table_owner*
[, *table_qualifier* [, *column_name*]]]

sp_columns

Returns information about the type of data that can be stored in one or more columns.

sp_columns *table_name* [, *table_owner*]
[, *table_qualifier*] [, *column_name*]

sp_databases

Returns a list of databases on a SQL Server.

sp_databases

sp_datatype_info

Returns information about a particular datatype or about all supported datatypes.

sp_datatype_info [*data_type*]

sp_fkeys

Returns logical foreign key information for the current database. Foreign keys must have been declared through the ANSI integrity constraint mechanism.

sp_fkeys *pktable_name* [, *pktable_owner*]
[, *pktable_qualifier*] [, *fktable_name*]
[, *fktable_owner*] [, *fktable_qualifier*]

sp_pkeys

Returns primary key information for a single table. Primary keys must have been declared through the ANSI integrity constraint mechanism.

sp_pkeys ***table_name*** **[,** ***table_owner*****]**
[, ***table_qualifier*****]**

sp_server_info

Returns a list of attribute names and matching values for SQL Server.

sp_server_info [***attribute_id*****]**

sp_special_columns

Returns the optimal set of columns that uniquely identify a row in a table or view; can also return a list of the columns that are automatically updated when any value in the row is updated by a transaction.

sp_special_columns ***table_name*** **[,** ***table_owner*****]**
[, ***table_qualifier*****] [,** ***col_type*****]**

sp_sproc_columns

Returns information about a stored procedure's input and return parameters.

sp_sproc_columns ***sp_name*** **[,** ***sp_owner*****]**
[, ***sp_qualifier*****] [,** ***column_name*****]**

sp_statistics

Returns a list of indexes on a single table.

sp_statistics ***table_name*** **[,** ***table_owner*****]**
[, ***table_qualifier*****] [,** ***index_name*****] [,** ***is_unique*****]**

sp_stored_procedures

Returns information about one or more stored procedures.

sp_stored_procedures [***sp_name*****] [,** ***sp_owner*****]**
[, ***sp_qualifier*****]**

sp_table_privileges

Returns privilege information for all columns in a table or view.

sp_table_privileges ***table_name*** **[,** ***table_owner***
[, ***table_qualifier*****]]**

sp_tables

Returns a list of objects that can appear in a **from** clause.

```
sp_tables [table_name] [, table_owner]
   [, table_qualifier][, table_type]
```

Error Messages and Message Numbers

Severity Levels 10 Through 18

Error messages with severity levels 10 through 16 are generated by problems caused by user errors, and can always be corrected by the user. Severity levels 17 and 18 do not terminate the user's session.

Error messages with severity levels 17 or higher should be reported to the System Administrator or Database Owner.

Severity Level	Explanation/Cause	Action
10: Status Information	Not an error. The SQL Server provides additional information after certain commands have been executed.	none
11: Specified Database Object Not Found	SQL Server can't find an object referenced in the command because the user has made a mistake in typing the name of a database object, because the user did not specify the object owner's name, or because of confusion about which database is current.	Check spelling of object names, use owner names if the object is not owned by you or "dbo", and make sure you're in the correct database.
12: Wrong Datatype Encountered	A problem with datatypes. For example, the user may have tried to enter a value of the wrong datatype into a column, or to compare columns of different (and incompatible) datatypes.	To correct comparison problems, use the **convert** function with **select**.

Severity Level	Explanation/Cause	Action
13: User Transaction Syntax Error	Something is wrong with the current user-defined transaction. For example, issuing a **commit** command without having issued a **begin transaction**, or trying to roll a transaction back to a savepoint that has not been defined (sometimes there may be a typing or spelling mistake in the name of the savepoint).	Severity level 13 can also indicate a deadlock, in which case the deadlock victim's process is rolled back. The user must restart his or her command.
14: Insufficient Permission to Execute Command	You don't have the permission necessary to execute the command or access the database object.	Ask the owner of the database object, the owner of the database, or the System Administrator to grant you permission to use the command or object in question.
15: Syntax Error in SQL Statements	The user has made a mistake in the syntax of the command. The text of these error messages includes the line numbers on which the mistake occurs, and the specific word near which it occurs.	Retype the command.
16: Miscellaneous User Error	The user has made some kind of non-fatal mistake that doesn't fall into any of the other categories.	Check command syntax and working database context.
17: Insufficient Resources	The command caused SQL Server to run out of resources (usually space for the database on the disk) or to exceed some limit set by the System Administrator. These system limits include the number of databases that can be open at the same time and the number of connections allowed to SQL Server. They are stored in system tables, and can be checked with the **sp_configure** command	The Database Owner can correct level 17 error messages indicating that you have run out of space. Other level 17 error messages should be corrected by the System Administrator.
18: Non-Fatal Internal Error Detected	Error messages with severity level 18 indicate some kind of internal software bug. However, the command runs to completion, and the connection to SQL Server is maintained.	

Severity Levels 19 Through 24

Fatal problems generate error messages with severity levels 19 and higher. They break the user's connection to SQL Server. To continue working, the user must restart the client program.

Severity Level	Explanation/Cause	Action
19: SQL Server Fatal Error in Resource	Some non-configurable internal limit has been exceeded, and SQL Server cannot recover gracefully.	Reconnect to SQL Server. See your System Administrator.
20: SQL Server Fatal Error in Current Process	SQL Server has encountered a bug in some command. The problem has affected only the current process; it is unlikely that the database itself has been damaged.	Run **dbcc** diagnostics. Reconnect to SQL Server. See your System Administrator.
21: SQL Server Fatal Error in Database Processes	SQL Server has encountered a bug that affects all the processes in the current database. However, it is unlikely that the database itself has been damaged.	Restart SQL Server and run the **dbcc** diagnostics. Reconnect to SQL Server. See your System Administrator.
22: SQL Server Fatal Error: Table Integrity Suspect	The table or index specified in the message has been damaged at some previous time by a software or hardware problem.	First, restart SQL Server and run **dbcc** to determine if other objects in the database are also damaged. If the problem is in the cache only, and not on the disk itself, restarting SQL Server will fix the problem. If restarting doesn't help, the problem is on the disk as well. Sometimes the problem can be solved by dropping the object specified in the error message.
23: SQL Server Fatal Error: Database Integrity Suspect	The integrity of the entire database is suspect due to damage caused at some previous time by a software or hardware problem.	Restart SQL Server and run the **dbcc** diagnostics. Even when the whole database is suspect, the damage may be confined to the cache, and the disk itself may be fine. If so, restarting SQL Server with **startserver** will fix the problem.
24: Hardware Error or System Table Corruption	Error messages with severity level 24 reflect some kind of media failure or (in rare cases) the corruption of *sysusages*.	The System Administrator may have to reload the database. It may be necessary to call your hardware vendor.