T-SQL Tutorial v1

2017

www.tsql.info

Backup and Restore

SQL Server allow to back up and restore your databases. Also, you can export and import security certificates and keys.

TSQL Backup

- Backup entire database to disk file
- Backup partial backup to disk file
- Backup database devices in a Striped Media Set

TSQL Restore

- Restore entire database
- Restore full and differential database backup

Backup entire database

The following example backs up the master database to a disk file.

BACKUP DATABASE Syntax:

```
BACKUP DATABASE { database_name }
TO backup_device [ ,...n ]
[ MIRROR TO clause ]
[ WITH { DIFFERENTIAL | [ ,...n ] } ];
```

BACKUP DATABASE Example:

```
BACKUP DATABASE master
TO DISK = 'E:\SQLServerBackup\master.bak'
WITH FORMAT;
GO
```

Messages:

```
Processed 464 pages for database 'master', file 'master' on file 1.
Processed 3 pages for database 'master', file 'mastlog' on file 1.
BACKUP DATABASE successfully processed 467 pages in 0.274 seconds (13.308 MB/sec).
```

Backup partial database

The following example backup partial the model database to a disk file.

BACKUP DATABASE Syntax:

```
BACKUP DATABASE { database_name }
READ_WRITE_FILEGROUPS [ , [ ,...n ] ]
TO backup_device [ ,...n ]
[ MIRROR TO clause ]
[ WITH { DIFFERENTIAL | [ ,...n ] } ];
```

Backup partial backup example:

```
BACKUP DATABASE model READ_WRITE_FILEGROUPS
TO DISK = 'E:\SQLServerBackup\model_partial.bak'
GO
```

Messages:

```
Processed 328 pages for database 'model', file 'modeldev' on file 1.
Processed 2 pages for database 'model', file 'modellog' on file 1.
BACKUP DATABASE...FILE=name successfully processed 330 pages in 0.325 seconds
(7.910 MB/sec).
```

Backup partial backup with differential example:

```
BACKUP DATABASE model READ_WRITE_FILEGROUPS
TO DISK = 'E:\SQLServerBackup\model_partial.dif'
WITH DIFFERENTIAL
GO
```

Messages:

```
Processed 40 pages for database 'model', file 'modeldev' on file 1.
Processed 2 pages for database 'model', file 'modellog' on file 1.
BACKUP DATABASE...FILE=name WITH DIFFERENTIAL successfully processed
42 pages in 0.147 seconds (2.182 MB/sec).
```

Backup database devices

Backup database devices in a Striped Media Set

The following example backup the model database devices in a striped media set.

Example:

```
BACKUP DATABASE model
TO DISK='E:\SQLServerBackup\1\model_1.bak',
DISK='E:\SQLServerBackup\2\model_2.bak'
WITH FORMAT,
MEDIANAME = 'modelStripedSet0',
MEDIADESCRIPTION = 'Striped media set for model database';
GO
```

Messages:

Processed 328 pages for database 'model', file 'modeldev' on file 1.
Processed 2 pages for database 'model', file 'modellog' on file 1.
BACKUP DATABASE successfully processed 330 pages in 0.350 seconds (7.345 MB/sec).

Restore entire database

Restore a full database backup from the logical backup device.

RESTORE DATABASE Syntax:

```
RESTORE DATABASE { database_name }
[ FROM [ ,...n ] ]
[ WITH
{
[ RECOVERY | NORECOVERY | STANDBY = {standby_file_name} ]
| , [ ,...n ]
| ,
| ,
| ,
| ,
| ,
```

} [,...n]
]
[;]

RESTORE DATABASE Example:

```
USE master;
GO
RESTORE DATABASE test_2
FROM test_2;
GO
```

Messages:

Processed 328 pages for database 'test_2', file 'test_2' on file 1.
Processed 2 pages for database 'test_2', file 'test_2_log' on file 1.
RESTORE DATABASE successfully processed 330 pages in 0.276 seconds (9.333 MB/sec).

Restore full and differential database backup

Restore a full database backup followed by a differential backup from backup device.

RESTORE DATABASE Syntax:

```
RESTORE DATABASE [ database_name ]
FROM DISK = [ backup_path ]
WITH FILE = [ file_number]
RECOVERY;
```

RESTORE DATABASE Example:

```
RESTORE DATABASE AdventureWorks2012 FROM DISK =
'Z:\SQLServerBackups\AdventureWorks2012.bak' WITH FILE = 9 RECOVERY;
```

Messages:

Processed 328 pages for database 'test_2', file 'test_2' on file 1.
Processed 2 pages for database 'test_2', file 'test_2_log' on file 1.
RESTORE DATABASE successfully processed 330 pages in 0.276 seconds (9.333

MB/sec).

Constraints

In the Constraints sections you can learn how to create a Primary Key Constraint or add a Foreign Key to a table. Also you can learn how to use commands to enable or disable keys.

Constraints operations

- Create a Primary Key
- Create a Foreign Key
- Disable a Foreign Key
- Enable a Foreign Key
- List table constraints
- Delete a Constraint Key

Create a Primary Key

To create a primary key in a table, use the command alter table with add constraint.

Departments table

```
USE tempdb;
GO
CREATE TABLE dbo.departments
(
id int NOT NULL,
name varchar(250)
 );
GO
```

Create Constraint Primary Key

```
USE tempdb;
GO
ALTER TABLE dbo.departments
ADD CONSTRAINT PK_DEP_ID
PRIMARY KEY CLUSTERED (ID);
GO
```

Create a Foreign Key

To Create a foreign key in an existing table, use the command alter table with add constraint.

Employees table

```
USE tempdb;
GO
CREATE TABLE dbo.EMPLOYEES(
ID INT NULL,
NAME VARCHAR (250) NULL,
JOB VARCHAR (30) NULL,
DEPT_ID INT NULL );
GO
```

Departments table

```
USE tempdb;
GO
CREATE TABLE dbo.departments(
id int NOT NULL,
name varchar(250) );
GO
```

Create Constraint Foreign Key

```
USE tempdb;
GO
ALTER TABLE dbo.EMPLOYEES
ADD CONSTRAINT
FK_DEPT_ID FOREIGN KEY(DEPT_ID)
REFERENCES dbo.DEPARTMENTS(ID);
GO
```

Disable a Foreign Key

To disable a foreign key constraint, use the command alter table with NOCHECK constraint.

Status Foreign Key

```
select name, type_desc, is_disabled from sys.foreign_keys;
```

name	type_desc	is_disabled
FK_DEPT_ID	FOREIGN_KEY_CONSTRAINT	0

Disable Foreign Key Constraint

```
USE tempdb;
GO
ALTER TABLE dbo.EMPLOYEES
NOCHECK CONSTRAINT FK_DEPT_ID;
GO
```

Status Foreign Key

```
select name, type_desc, is_disabled from sys.foreign_keys;
```

name	type_desc	is_disabled
FK_DEPT_ID	FOREIGN_KEY_CONSTRAINT	1

Enable a Foreign Key

To Enable a foreign key constraint, use the command alter table with CHECK constraint.

Status Foreign Key

```
select name, type_desc, is_disabled from sys.foreign_keys;
```

name	type_desc	is_disabled
FK_DEPT_ID	FOREIGN_KEY_CONSTRAINT	1

Enable Foreign Key Constraint

```
USE tempdb;
GO
ALTER TABLE dbo.EMPLOYEES
CHECK CONSTRAINT FK_DEPT_ID;
GO
```

Status Foreign Key

select name, type_desc, is_disabled from sys.foreign_keys;

name	type_desc	is_disabled
FK_DEPT_ID	FOREIGN_KEY_CONSTRAINT	0

List table constraints

To list table constraints, use select on sys.foreign_keys, sys.key_constraints or sys.all_objects.

sys.foreign_keys

SELECT * FROM sys.foreign_keys;

sys.key_constraints

SELECT * FROM sys.key_constraints;

sys.all_objects

SELECT * FROM sys.all_objects
WHERE TYPE IN ('F','PK')
ORDER BY create_date DESC;

Delete a Constraint Key

To delete a constraint key from an existing table, use the command alter table with drop constraint.

Create Constraint Foreign Key

USE tempdb;
GO
ALTER TABLE dbo.EMPLOYEES
ADD CONSTRAINT FK_DEPT_ID FOREIGN KEY(DEPT_ID)
REFERENCES dbo.DEPARTMENTS(ID);
GO

Delete Constraint Foreign Key

```
USE tempdb;
GO
ALTER TABLE dbo.EMPLOYEES
DROP CONSTRAINT FK_DEPT_ID;
GO
```

Cursors

In this chapter you can learn how to work with cursors using operations like declare cursor, create procedure, fetch, delete, update, close, set, deallocate.

Cursor operations

- Declare cursor
- Create procedure
- Open cursor
- Close cursor
- Fetch cursor
- Deallocate cursor
- Delete
- Update

Declare cursors

Declare cursor Syntax:

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_query_statement
[ FOR UPDATE [ OF column_name [ ,...n ] ] ] ;
```

Declare simple cursor example:

```
DECLARE product_cursor CURSOR
FOR SELECT * FROM model.dbo.products;
OPEN product_cursor
FETCH NEXT FROM product_cursor;
```

Create procedure

Create procedure example:

```
USE model;
GO
IF OBJECT_ID ( 'dbo.productProc', 'P' ) IS NOT NULL
 DROP PROCEDURE dbo.productProc;
GO
CREATE PROCEDURE dbo.productProc
 @varCursor CURSOR VARYING OUTPUT
AS
 SET NOCOUNT ON;
 SET @varCursor = CURSOR
 FORWARD_ONLY STATIC FOR
 SELECT product_id, product_name
 FROM dbo.products;
 OPEN @varCursor;
GO
```

Open cursors

Open cursor Syntax:

```
OPEN { { cursor_name } | cursor_variable_name }
```

Open cursor example:

```
USE model;
GO
DECLARE Student_Cursor CURSOR FOR
SELECT id, first_name, last_name, country
FROM dbo.students WHERE country != 'US';
OPEN Student_Cursor;
FETCH NEXT FROM Student_Cursor;
WHILE @@FETCH_STATUS = 0
 BEGIN
 FETCH NEXT FROM Student_Cursor;
 END;
CLOSE Student_Cursor;
```

```
DEALLOCATE Student_Cursor;
GO
```

Close cursors

Close cursor Syntax:

```
CLOSE { { cursor_name } | cursor_variable_name }
```

Close cursor example:

```
USE model;
GO
DECLARE Student_Cursor CURSOR FOR
SELECT ID, FIRST_NAME FROM dbo.students;
OPEN Student_Cursor;
FETCH NEXT FROM Student_Cursor;
WHILE @@FETCH_STATUS = 0
 BEGIN
 FETCH NEXT FROM Student_Cursor;
 END;
CLOSE Student_Cursor;
DEALLOCATE Student_Cursor;
GO
```

Fetch cursors

Fetch cursor Syntax:

```
FETCH
[ NEXT | PRIOR | FIRST | LAST
| ABSOLUTE { n | @nvar }
| RELATIVE { n | @nvar }
]
FROM
{ { cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ] ;
```

Fetch in cursors example:

```
USE model;
GO
DECLARE Student_Cursor CURSOR FOR
SELECT id, first_name, last_name, country
FROM dbo.students WHERE country = 'US';
OPEN Student_Cursor;
FETCH NEXT FROM Student_Cursor;
WHILE @@FETCH_STATUS = 0
 BEGIN
 FETCH NEXT FROM Student_Cursor;
 END;
CLOSE Student_Cursor;
DEALLOCATE Student_Cursor;
GO
```

Deallocate cursors

When the cursor is deallocated, the data structures comprising the cursor are released by Microsoft SQL Server.

Deallocate cursor Syntax:

```
DEALLOCATE { { cursor_name } | cursor_variable_name }
```

Deallocate cursor example:

```
USE model;
GO
DECLARE @CursorName CURSOR
SET @CursorName = CURSOR LOCAL SCROLL FOR
SELECT * FROM dbo.students;

DEALLOCATE @CursorName;

SET @CursorName = CURSOR LOCAL SCROLL FOR
SELECT * FROM dbo.students;
GO
```

Delete in cursors

Delete in cursors example:

```
USE model;
GO
DECLARE MyCursor CURSOR FOR
 SELECT *
 FROM dbo.students
 WHERE first_name = 'David' AND last_name = 'BROWN' AND id = 6 ;
OPEN MyCursor;
FETCH FROM MyCursor;
DELETE FROM dbo.students WHERE CURRENT OF MyCursor;
CLOSE MyCursor;
DEALLOCATE MyCursor;
GO
```

Update in cursors

Update in cursors example:

```
USE model;
GO
DECLARE test_cursor CURSOR LOCAL FOR
 SELECT id, first_name, last_name, section
 FROM dbo.students WHERE id = 2;
OPEN test_cursor;
FETCH test_cursor;
UPDATE dbo.students
SET section = 'Medicine'
FROM dbo.students
WHERE CURRENT OF test_cursor;
GO

SELECT id, first_name, last_name, section FROM dbo.students;
GO
```

Data Types

The Transact SQL language allow you to use various data types like: Numeric (int, numeric, decimal, float), Character Strings (char, varchar), Unicode Character Strings (nchar, nvarchar) , Date (date, datetime, datetime2, time) and other data types.

Binary Strings

- Binary
- Varbinary

Character Strings

- Char
- Varchar

Date and Time

- Date
- Datetime
- Datetime2
- Datetimeoffset
- Smalldatetime
- Time

Numerics

- Bigint
- Int
- Smallint
- Tinyint
- Decimal
- Numeric

Unicode Character Strings

- Nchar
- Nvarchar

Other Data Types

- Rowversion
- Uniqueidentifier
- Table

Binary

On Transact SQL language the binary is part of binary strings data types and have fixed length.
The string length must be a value from 1 through 8,000.

Binary syntax:

binary [(n)]

Binary example:

```
USE model;
GO
DECLARE @myVar BINARY(2);
SET @myVar = 12345678;
SET @myVar = @myVar + 2;
SELECT CAST( @myVar AS INT);
GO
```

Results

24912

Varbinary

On Transact SQL language the binary is part of binary strings data types and have variable length. The string length must be a value from 1 through 8,000.

Varbinary syntax:

varbinary [(n)]
varbinary [(max)]

Range	Storage
2^31-1 bytes (2 GB)	2 Bytes

Varbinary example:

```
USE model;
GO
DECLARE @myVar VARBINARY(2);
SET @myVar = 123456789;
SET @myVar = @myVar + 3;
SELECT CAST( @myVar AS INT);
GO
```

Results

```
52504
```

Char

On Transact SQL language the char is part of character strings data types and have fixed length.
The string length must be a value from 1 through 8,000.

Char syntax:

```
char [ ( n ) ]
```

Char example:

```
USE model;
GO
CREATE TABLE myCharTable ( a char(25) );
GO
INSERT INTO myCharTable VALUES ('abc + def');
GO
SELECT a FROM myCharTable;
GO
```

Results

```
abc + def
```

Varchar

On Transact SQL language the varchar is part of character strings data types and have variable length. The string length must be a value from 1 through 8,000.

Varchar syntax:

varchar [(n)]
varchar [(max)]

Varchar example:

```
USE model;
GO
CREATE TABLE varcharTable ( a varchar(10) );
GO
INSERT INTO varcharTable VALUES ('abcdefghij');
GO
SELECT a FROM varcharTable;
GO
```

Results

abcdefghij

```
USE model;
GO
DECLARE @myVar AS varchar(20) = 'abc123';
SELECT @myVar as 'My column', DATALENGTH(@myVar) as 'Length';
GO
```

My column	Length
abc123	6

Date

On Transact SQL language the date is part of date and time data types and define a date on sql server.

Date syntax:

date

Property	Value
Default string literal format	YYYY-MM-DD
Range	0001-01-01 through 9999-12-31
Length	10
Storage size	3 bytes, fixed
Calendar	Gregorian

Date example:

```
USE model;
GO
DECLARE @date date= '08-21-14';
SELECT @date AS 'Date';
GO
```

Date
2014-08-21

Datetime

On Transact SQL language the datetime is part of date and time data types and define a date that is combined with a time of day with fractional seconds.

Datetime syntax:

datetime

Property	Value
Range	January 1, 1753, through December 31, 9999
Length	19 minimum - 23 maximum
Storage size	8 bytes
Calendar	Gregorian

Datetime example:

```
USE model;
GO
DECLARE @date date= '08-21-14';
DECLARE @datetime datetime = @date;
SELECT @datetime AS 'Datetime', @date AS 'Date';
GO
```

Datetime	Date
2014-08-21 00:00:00.000	2014-08-21

Datetime2

On Transact SQL language the datetime2 is part of date and time data types and is an extension of the datetime type that has a larger date range, a larger default fractional precision.

Datetime2 syntax:

datetime2

Property	Value
Default string literal format	YYYY-MM-DD hh:mm:ss[.fractional seconds]
Range	0001-01-01 through 9999-12-31
Length	19 minimum - 27 maximum
Storage size	8 bytes
Calendar	Gregorian

Datetime2 example:

```
USE model;
GO
DECLARE @datetime2 datetime2 = '08-21-14 10:09:30.123';
SELECT @datetime2 AS 'Datetime2';
GO
```

Datetime2

2014-08-21 10:09:30.1230000

Datetimeoffset

On Transact SQL language the datetimeoffset is part of date and time data types and define a date that is combined with a time of a day that has time zone awareness.

Datetimeoffset syntax:

datetimeoffset [(fractional seconds precision)]

Property	Value
Default string literal format	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [{+|-}hh:mm]
Range	0001-01-01 - 9999-12-31
Length	26 minimum - 34 maximum
Storage size	10 bytes

Datetimeoffset example:

```
USE model;
GO
DECLARE @datetimeoffset datetimeoffset(3) = '2014-08-21 10:49:32.1234 +10:0';
DECLARE @datetime2 datetime2(3)=@datetimeoffset;
SELECT @datetimeoffset AS 'Datetimeoffset', @datetime2 AS 'Datetime2';
GO
```

Datetimeoffset	Datetime2
2014-08-21 10:49:32.123 +10:00	2014-08-21 10:49:32.123

Smalldatetime

On Transact SQL language the smalldatetime is part of date and time data types and define a date that is combined with a time of day.

Smalldatetime syntax:

smalldatetime

Property	Value
Range	1900-01-01 through 2079-06-06
Length	19 maximum
Storage size	4 bytes
Calendar	Gregorian

Smalldatetime example:

```
USE model;
GO
DECLARE @smalldatetime smalldatetime = '2014-08-21 11:03:17';
DECLARE @date date = @smalldatetime
SELECT @smalldatetime AS 'Smalldatetime', @date AS 'Date';
GO
```

Smalldatetime	Date
2014-08-21 11:03:00	2014-08-21

Time

On Transact SQL language the time is part of date and time data types and define a time of a day.

Time syntax:

time

Property	Value
Default string literal format	hh:mm:ss[.nnnnnnn]
Range	00:00:00.0000000 - 23:59:59.9999999
Length	8 minimum - 16 maximum
Storage size	5 bytes

Time example:

```
USE model;
GO
DECLARE @time time = '08-21-14 10:21:12.123';
SELECT @time AS 'Time';
GO
```

Time
10:21:12.1230000

Bigint

On Transact SQL language the bigint is an numeric data type. On this page you can read about max value and find an simple example.

Bigint syntax:

Range	Storage
-2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)	8 Bytes

Bigint example:

```
USE model;
GO
CREATE TABLE test_table ( a bigint );
GO
INSERT INTO test_table VALUES (9223372036854775807);
GO
SELECT a FROM test_table;
GO
```

Results
9223372036854775807

Int

On Transact SQL language the int is an numeric data type. The int data type is the primary integer data type in SQL Server.

Int syntax:

Range	Storage
-2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)	4 Bytes

Int example:

```
USE model;
GO
CREATE TABLE myIntTable ( b int );
GO
INSERT INTO myIntTable VALUES (214483647);
GO
SELECT b FROM myIntTable;
GO
```

Results

```
214483647
```

Smallint

On Transact SQL language the smallint is an numeric data type. Here you can read about max value and find an simple example.

Smallint syntax:

Range	Storage
-2^15 (-32,768) to 2^15-1 (32,767)	2 Bytes

Smallint example:

```
USE model;
GO
CREATE TABLE mySmallintTable ( c smallint );
GO
INSERT INTO mySmallintTable VALUES (32767);
GO
SELECT c FROM mySmallintTable;
GO
```

Results

```
32767
```

Tinyint

On Transact SQL language the tinyint is an numeric data type. The maximum value of tinyint data type is 255.

Tinyint syntax:

Range	Storage
0 to 255	1 Byte

Tinyint example:

```
USE model;
GO
CREATE TABLE myTinyintTable ( d tinyint );
GO
INSERT INTO myTinyintTable VALUES (255);
GO
SELECT d FROM myTinyintTable;
GO
```

Results

255

Decimal

On Transact SQL language the decimal is the same like numeric data types and have fixed precision and scale.

Decimal syntax:

Precision	Storage
1 - 9	5 Bytes
10-19	9 Bytes

20-28	13 Bytes
29-38	17 Bytes

Decimal example:

```
USE model;
GO
CREATE TABLE myDecTable ( b decimal (7,2) );
GO
INSERT INTO myDecTable VALUES (234);
GO
SELECT b FROM myDecTable;
GO
```

Results

234.00

Numeric

On Transact SQL language the numeric data types that have fixed precision and scale.

Numeric syntax:

Precision	Storage
1 - 9	5 Bytes
10-19	9 Bytes
20-28	13 Bytes
29-38	17 Bytes

Numeric example:

```
USE model;
GO
CREATE TABLE myNumTable ( a numeric (12,6) );
GO
INSERT INTO myNumTable VALUES (777.123);
GO
SELECT a FROM myNumTable;
GO
```

Results

777.123000

Nchar

On Transact SQL language the nchar is part of unicode character strings data types and have fixed length. The string length must be a value from 1 through 4,000.

Nchar syntax:

nchar [(n)]

Nchar example:

```
USE model;
GO
CREATE TABLE myNcharTable ( a nchar(20) );
GO
INSERT INTO myNcharTable VALUES ('This is an example');
GO
SELECT a FROM myNcharTable;
GO
```

Results

This is an
example

Nvarchar

On Transact SQL language the nvarchar is part of unicode character strings data types and have variable length. The string length must be a value from 1 through 4,000.

Nvarchar syntax:

nvarchar [(n)]
nvarchar [(max)]

Range	Storage
2^31-1 bytes (2 GB)	2 Bytes

Nvarchar example:

```
USE model;
GO
CREATE TABLE nvarcharTable ( a nvarchar(25) );
GO
INSERT INTO nvarcharTable VALUES ('This is an example');
GO
SELECT a FROM nvarcharTable;
GO
```

Results

```
This is an
example
```

Rowversion

On Transact SQL language the rowversion is a data type that generate automatically unique binary numbers within a database. The storage size is 8 bytes.

Rowversion syntax:

```
rowversion
```

Rowversion example:

```
USE model;
GO
CREATE TABLE myTest (id int PRIMARY KEY, name char(20), test_column
rowversion);
GO
INSERT INTO myTest (id, name) VALUES (1, 'test_1');
GO
INSERT INTO myTest (id, name) VALUES (2, 'test_2');
GO
SELECT * FROM myTest;
GO
```

id	name	test_column
1	test_1	0x0000000000000FA1
2	test_2	0x0000000000000FA2

Uniqueidentifier

On Transact SQL language the uniqueidentifier is a character type for the purposes of conversion from a character expression.

Uniqueidentifier syntax:

uniqueidentifier

Uniqueidentifier example:

```
USE model;
GO
DECLARE @id_var uniqueidentifier = NEWID();
SELECT @id_var AS 'Result';
GO
```

Result
4BCF38AD-BB98-42CA-82D3-97DBE081EB4B

Table

On Transact SQL language the table variable is a special data type that can be used to store a result set for processing at a later time. Table variables are automatically will be emptied at the end of the function or stored procedure in which they are defined.

Table syntax:

table

Table example:

```
USE model;
GO
DECLARE @TableVar table(id int NOT NULL, OldValue char(2), NewValue char(2));
UPDATE my_table SET b='c'
```

```
OUTPUT INSERTED.a, DELETED.b, INSERTED.b
INTO @TableVar;
SELECT id, OldValue, NewValue FROM @TableVar;
GO
```

id	OldValue	NewValue
1	a	c
2	b	c

Operators

Operators

Arithmetic
Bitwise
Comparison
Compound
Logical

Arithmetic

Operator	Meaning
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Bitwise

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Comparison

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Compound

Operator	Meaning
+=	Add Equals
-=	Subtract Equals
*=	Multiply Equals
/=	Divide Equals
%=	Modulo Equals
&=	Bitwise AND Equals
^-=	Bitwise Exclusive Equals
|*=	Bitwise OR Equals

Logical

Operator	Meaning
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if both expressions are TRUE.
ANY	TRUE if any one of a set of comparisons are TRUE.
BETWEEN	TRUE if the operand is within the range of comparisons.
EXISTS	TRUE if a subquery contains any rows.
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other operator.

OR	TRUE if either expression is TRUE.
SOME	TRUE if some of a set of comparisons are TRUE.

Select Query

Select query

Group By
Having
Order By
Where
Between
Like
Exists
In
Some
Any

Group By

Group By is used when we have an aggregate function(like: count, max, min, sum, avg) in our query. In first example the order is random, in second example we use the column amount for the order.

	CONTRACT_ID	AMOUNT
1		400
2		500
3		700
4		500
1		400
5		200

Group By Example 1:

```
SELECT c.contract_id, c.amount, COUNT(c.contract_id) AS CtrCount
 FROM contracts c
GROUP BY c.contract_id, c.amount;
```

	Contract_Id	Amount	CtrCount
5		200	1

1	400	2
2	500	1
4	500	1
3	700	1

Group By Example 2:

```
SELECT c.contract_id, c.amount, COUNT(*) AS CtrCount
FROM contracts c
GROUP BY c.amount, c.contract_id
ORDER BY c.amount;
```

Contract_Id	Amount	CtrCount
5	200	1
1	400	2
2	500	1
4	500	1
3	700	1

Having

Having Example:

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
SELECT c.customer_id, c.amount
FROM contracts c
WHERE c.amount < 2500
GROUP BY c.customer_id, c.amount
HAVING MIN(c.amount) > 1000 ;
```

Customer_Id	Amount
2	1200
3	2000
2	1500

Order By

Order By Example:

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
SELECT c.customer_id, c.amount
FROM contracts c
WHERE c.amount < 2500
GROUP BY c.customer_id, c.amount
HAVING MIN(c.amount) > 1000
ORDER BY c.amount ;
```

Customer_Id	Amount
2	1200
2	1500
3	2000

```
SELECT c.customer_id, c.amount
FROM contracts c
WHERE c.amount < 2500
GROUP BY c.customer_id, c.amount
HAVING MIN(c.amount) > 1000
ORDER BY c.amount DESC;
```

Customer_Id	Amount
3	2000

2	1500
2	1200

Where

Where Example:

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
SELECT c.contract_id, c.customer_id, c.amount
FROM contracts c
WHERE c.amount > 900 ;
```

Contract_Id	Customer_Id	Amount
4	1	1000
5	2	1200
7	3	2000
8	2	1500

Between operator

Between operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas
5	San Diego	California

Between Example 1:

Find the cities with id between 2 and 4.
SELECT * FROM cities WHERE city_id BETWEEN 2 AND 4;

Between Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas

Insurance table:

ID	TYPE	START_DATE	END_DATE
2	Life	2012-01-21	2052-01-20
3	Health	2013-01-26	2018-01-25
4	Vehicle	2010-01-23	2012-01-22
5	Vehicle	2008-06-21	2016-06-20
6	Health	2009-03-07	2030-03-06

Between Example 2:

Find the insurance policies underwritten between '01-JAN-2010' and '31-JAN-2013'.
SELECT * FROM insurance WHERE start_date BETWEEN '2010-01-01' AND '2013-01-31';

Between Result:

ID	TYPE	START_DATE	END_DATE

2	Life	2012-01-21	2052-01-20
3	Health	2013-01-26	2018-01-25
4	Vehicle	2010-01-23	2012-01-22

Like operator

Like operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas
5	San Diego	California

Like Example 1:

Find the city name that contain letters: an.
SELECT * FROM cities WHERE name LIKE '%an%';

Like Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
4	San Antonio	Texas
5	San Diego	California

Like Example 2:

Find the cities name that start with: Sa.
SELECT * FROM cities WHERE name LIKE '%Sa%';

Like Result:

CITY_ID	NAME	STATE
4	San Antonio	Texas

5	San Diego	California

Like Example 3:

Find the cities name that end with: go.
SELECT * FROM cities WHERE name LIKE '%go';

Like Result:

CITY_ID	NAME	STATE
3	Chicago	Illinois
5	San Diego	California

Exists operator

Exists operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas
5	San Diego	California

States table:

STATE_ID	NAME
1	Arizona
2	California
3	Texas
4	Michigan

Exists Example:

Find the cities that have the column state correspondent in the states table.
SELECT * FROM cities c WHERE EXISTS (SELECT * FROM states s WHERE c.state=s.name);

Exists Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
4	San Antonio	Texas
5	San Diego	California

NOT Exists Example:

Find the cities that NOT have the column state correspondent in the states table.
SELECT * FROM cities c WHERE NOT EXISTS (SELECT * FROM states s WHERE c.state=s.name);

NOT Exists Result:

CITY_ID	NAME	STATE
1	New York	New York
3	Chicago	Illinois

IN operator

IN operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas

5	San Diego	California

States table:

STATE_ID	NAME
1	Arizona
2	California
3	Texas
4	Michigan

IN Example:

Find the cities that have the state in the states table.
SELECT * FROM cities c WHERE c.state IN (SELECT s.name FROM states s);

IN Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
4	San Antonio	Texas
5	San Diego	California

NOT IN Example:

Find the cities that NOT have the state in the states table.
SELECT * FROM cities c WHERE c.state NOT IN (SELECT s.name FROM states s);

NOT IN Result:

CITY_ID	NAME	STATE
1	New York	New York
3	Chicago	Illinois

Some operator

Some operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas
5	San Diego	California

States table:

STATE_ID	NAME
1	Arizona
2	California
3	Texas
4	Michigan

Some Example:

Find the cities that have the state in the states table using = some.
SELECT * FROM cities c WHERE c.state = SOME (SELECT s.name FROM states s);

Some Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
4	San Antonio	Texas
5	San Diego	California

Any operator

Any operator:

Cities table:

CITY_ID	NAME	STATE
1	New York	New York
2	Los Angeles	California
3	Chicago	Illinois
4	San Antonio	Texas

5	San Diego	California

States table:

STATE_ID	NAME
1	Arizona
2	California
3	Texas
4	Michigan

Any Example:

Find the cities that have the any state in the states table using = any.
SELECT * FROM cities c WHERE c.state = ANY (SELECT s.name FROM states s);

Any Result:

CITY_ID	NAME	STATE
2	Los Angeles	California
4	San Antonio	Texas
5	San Diego	California

SET Statements

On Transact sql language the SET statements allow you to change the current session handling of specific information like: dateformat, system language, lock timeout, rowcount.

Date and time

- SET Datefirst
- SET Dateformat

Locks

- SET Deadlock_priority
- SET Lock_timeout

Miscellaneous

- SET Concat_null_yields_null
- SET Cursor_close_on_commit
- SET Identity_insert
- SET Language

Query Execution

- SET Rowcount
- SET Noexec

SET Datefirst

SET Datefirst - sets the first day of the week to a number from 1 through 7.

SET Datefirst Syntax:

SET DATEFIRST { number | @number_variable } ;

SET Datefirst Example:

SET DATEFIRST 1 ;

Messages:
Command(s) completed successfully.

SELECT @@DATEFIRST AS 'First Day';

Result:
1

SET Dateformat

SET Dateformat - sets the order of the month, day, and year date parts.

SET Dateformat Syntax:

SET DATEFORMAT { format | @format_variable } ;

SET Dateformat Example:

SET DATEFORMAT dmy;
GO
DECLARE @date_variable datetime2 = '31/08/2014 10:11:43.1234567';
SELECT @date_variable;
GO

Result:
2014-08-31 10:11:43.1234567

SET Deadlock_priority

SET Deadlock_priority - sets the importance of the current session if it is deadlocked with another session.

SET Deadlock_priority Syntax:

SET DEADLOCK_PRIORITY { LOW | NORMAL | HIGH | <numeric-priority> | @deadlock_variable } ;
<numeric-priority> ::= { -10 | -9 | ... | 0 | ... | 9 | 10 }

SET Deadlock_priority Example:

```
SET DEADLOCK_PRIORITY NORMAL;
GO
```

Messages:

```
Command(s) completed
successfully.
```

SET Lock_timeout

SET Lock_timeout - sets the number of milliseconds of statement that waits for a lock to be released.

SET Lock_timeout Syntax:

```
SET LOCK_TIMEOUT milliseconds_number ;
```

SET Lock_timeout Example:

```
SET LOCK_TIMEOUT 3600;
GO
```

Messages:

```
Command(s) completed
successfully.
```

SET Concat_null_yields_null

SET Concat_null_yields_null - Checks whether concatenation results are treated as null or empty string values.

SET Concat_null_yields_null Syntax:

```
SET CONCAT_NULL_YIELDS_NULL { ON | OFF } ;
```

SET Concat_null_yields_null Example:

```
USE model;
GO
SET CONCAT_NULL_YIELDS_NULL ON;
GO
SELECT 'test' + NULL;
GO
```

Results

NULL

```
USE model;
GO
SET CONCAT_NULL_YIELDS_NULL OFF;
GO
SELECT 'test' + NULL;
GO
```

Results

test

SET Cursor_close_on_commit

SET Cursor_close_on_commit - The default value for CURSOR_CLOSE_ON_COMMIT is OFF.
With CURSOR_CLOSE_ON_COMMIT set OFF the server will not close cursors when you commit a transaction.

SET Cursor_close_on_commit Syntax:

```
SET CURSOR_CLOSE_ON_COMMIT { ON | OFF } ;
```

SET Cursor_close_on_commit Example:

```
USE model;
GO
CREATE TABLE my_table (a INT, b CHAR(10));
GO
INSERT INTO my_table VALUES (1,'a'), (2,'b');
GO
```

```
SET CURSOR_CLOSE_ON_COMMIT OFF;
GO
PRINT 'BEGIN TRANSACTION';
BEGIN TRAN;
PRINT 'Declare cursor';
DECLARE my_cursor CURSOR FOR SELECT * FROM my_table;
PRINT 'Open cursor';
OPEN my_cursor;
PRINT 'COMMIT TRANSACTION';
COMMIT TRAN;
PRINT 'Use cursor after commit transaction';
FETCH NEXT FROM my_cursor;
CLOSE my_cursor;
DEALLOCATE my_cursor;
GO
```

Messages

BEGIN TRANSACTION
Declare cursor
Open cursor
COMMIT TRANSACTION
Use cursor after commit transaction

SET Identity_insert

SET Identity_insert - allow to be inserted explicit values into the identity column of a table.

SET Identity_insert Syntax:

```
SET IDENTITY_INSERT [ database_name . [ schema_name ] . ] table { ON | OFF } ;
```

SET Identity_insert Example:

```
USE model;
GO
CREATE TABLE Department(
ID INT IDENTITY NOT NULL PRIMARY KEY, Name VARCHAR(250) NOT NULL);

GO
INSERT INTO Department(Name)
```

VALUES ('Anthropology'), ('Biology'), ('Chemistry'), ('Computer Science'), ('Economics');
GO
DELETE FROM Department WHERE name='Biology';
GO
SELECT * FROM Departments;
GO

ID	Name
1	Anthropology
3	Chemistry
4	Computer Science
5	Economics

USE model;
GO
INSERT INTO Departments (ID, Name) VALUES (2, 'Biology');
GO

Messages

Msg 544, Level 16, State 1, Line 1
Cannot insert explicit value for identity column in table 'Departments' when IDENTITY_INSERT is set to OFF.

USE model;
GO
SET IDENTITY_INSERT Departments ON;
GO

Messages

Command(s) completed
successfully.

USE model;
GO
INSERT INTO Departments (ID, Name) VALUES (2, 'Biology');
GO

Messages

(1 row(s) affected)

SET Language

SET Language - sets the language of session.
The session language establish the format of date and system messages.

SET Language Syntax:

SET LANGUAGE { [N] 'language' | @language_variable } ;

SET Language Example:

```
USE model;
GO
DECLARE @MyDay DATETIME;
SET @MyDay = '06/21/2014';

SET LANGUAGE French;
SELECT DATENAME(month, @MyDay) AS 'French Month';

SET LANGUAGE English;
SELECT DATENAME(month, @MyDay) AS 'English Month' ;
GO
```

French Month
Juin
English Month
June

SET Rowcount

SET Rowcount - sets the number of rows for sql query.
When the specified number of rows are returned the execution of query stops.

SET Rowcount Syntax:

SET ROWCOUNT { number | @number_variable } ;

SET Rowcount Example:

```
USE model;
GO
SET ROWCOUNT 3;
```

```
GO
SELECT * FROM Departments WHERE id <=3;
GO
```

ID	Name
1	Anthropology
2	Biology
3	Chemistry

```
USE model;
GO
SET ROWCOUNT 2;
GO
SELECT * FROM Departments;
GO
```

ID	Name
1	Anthropology
2	Biology

SET Noexec

SET Noexec - sets the compile of each query but does not execute the queries.

SET Noexec Syntax:

```
SET NOEXEC { ON | OFF } ;
```

SET Noexec Example:

```
USE model;
GO
PRINT 'OK';
GO
SET NOEXEC ON;
GO
SELECT * FROM Departments WHERE id > 3;
GO
SET NOEXEC OFF;
GO
```

Messages

OK

SQL Tutorial

Select

Select Syntax:

SELECT column(s)FROM table ;
SELECT * FROM table ;

Store table:

OBJECT_ID	PRICE
1	100
2	300
3	800
4	300

Example 1:

SELECT * FROM store;

OBJECT_ID	PRICE
1	100
2	300
3	800
4	300

Example 2:

SELECT price FROM store;

PRICE
100
300
800
300

Distinct

Distinct Syntax:

SELECT DISTINCT column_name FROM table_name ;

Store table:

OBJECT_ID	PRICE
1	100
2	400
3	800
4	400

Example:

SELECT DISTINCT price FROM store;

PRICE
100
400
800

Where

Where Syntax:

SELECT column_name(s) FROM table_name WHERE condition ;

Store table:

OBJECT_ID	PRICE
1	200
2	500
3	900
4	500

Example 1:

SELECT * FROM store WHERE price = 500;

OBJECT_ID	PRICE
2	500

4	500

Example 2:

SELECT * FROM store WHERE price > 500;

OBJECT_ID	PRICE
3	900

And & OR

Store table:

OBJECT_ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Example 1:

SELECT * FROM store WHERE name='B' **AND** price = 500;

OBJECT_ID	PRICE	NAME
2	500	B

Example 2:

SELECT * FROM store WHERE name='B' **OR** price = 500;

OBJECT_ID	PRICE	NAME
2	500	B
4	500	D

Example 3:

SELECT * FROM store WHERE price = 900 **AND** (name='A' **OR** name='C');

OBJECT_ID	PRICE	NAME

3	900	C

Order By

Order By Syntax:

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

Store table:

OBJECT_ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Example 1:

SELECT * FROM store ORDER BY price, name;

OBJECT_ID	PRICE	NAME
1	200	A
2	500	B
4	500	D
3	900	C

Example 2:

SELECT * FROM store ORDER BY name DESC;

OBJECT_ID	PRICE	NAME
4	500	D
3	900	C
2	500	B
1	200	A

Group By

Group By Syntax:

SELECT column_name1, aggregate_function(column_name2)

FROM table GROUP BY column_name1

Store table:

OBJECT_ID	PRICE	TYPE
1	200	LOW
2	500	MEDIUM
3	900	HIGH
4	500	MEDIUM

Example:

SELECT type, SUM(price) FROM store GROUP BY type;

TYPE	PRICE
LOW	200
MEDIUM	1000
HIGH	900

Having

Having Syntax:

SELECT column_name(s), aggregate_function(column_name)
FROM my_table
WHERE condition {optional}
GROUP BY column_name(s)
HAVING (aggregate_function condition)

Having Example:

Sales table:

ID	PRICE	CUSTOMER
1	200	David
2	500	Linda
3	900	Tom
4	500	David
5	1200	Ellen
6	1200	Linda

```
SELECT customer, SUM(price)
FROM sales
GROUP BY customer
HAVING SUM(price) > 1000
```

Having Result:

customer	SUM(price)
Linda	1700
Ellen	1200

Like

Like Syntax:

SELECT column(s) FROM table WHERE column LIKE pattern

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	John	21
2	Samantha	22

3	Tom	23
4	James	24
5	Sandra	24

Like Example 1:

Find the employee names that contain letters: am.
SELECT * FROM employee WHERE name LIKE '%am%';

Like Result:

EMPLOYEE_ID	NAME	DEP_ID
2	Samantha	22
4	James	24

Like Example 2:

Find the employee names that begin with: J.
SELECT * FROM employee WHERE name LIKE 'J%';

Like Result:

EMPLOYEE_ID	NAME	DEP_ID
1	John	21
4	James	24

Like Example 3:

Find the employee names that end with: a.
SELECT * FROM employee WHERE name LIKE '%a';

Like Result:

EMPLOYEE_ID	NAME	DEP_ID
2	Samantha	22
5	Sandra	24

Insert into

Insert into Syntax:

INSERT INTO table_name VALUES (value1, value2, ...)

OR

INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...)

Store table:

	OBJECT_ID	PRICE	NAME
1		200	A
2		500	B
3		900	C
4		500	D

Example 1:

INSERT INTO store VALUES (5, 600, 'E');

Example 2:

INSERT INTO store(object_id, price, name) VALUES (6, 400, 'F');

	OBJECT_ID	PRICE	NAME
1		200	A
2		500	B
3		900	C
4		500	D
5		600	E
6		400	F

Update

Update Syntax:

UPDATE table_name SET column1 = new_value1, column2 = new_value2,... WHERE {condition}

IF you don't put the {condition} then all records on the updated column will be chenged.

Store table:

OBJECT_ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Example 1:

```
UPDATE store SET price = 300 WHERE object_id=1 AND name='A';

SELECT * FROM store WHERE object_id=1 AND name='A';
```

OBJECT_ID	PRICE	NAME
1	300	A

Example 2:

```
UPDATE store SET price = 1000, name = 'Y' WHERE object_id=3;

SELECT * FROM store WHERE object_id=3;
```

OBJECT_ID	PRICE	NAME
3	1000	Y

Delete

Delete Syntax:

```
DELETE FROM table_name WHERE {condition}
```

IF you don't put the {condition} then all the records from the table will be erased.

Store table:

OBJECT_ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Example:

DELETE FROM store WHERE price=500;

SELECT * FROM store;

OBJECT_ID	PRICE	NAME
1	200	A
3	900	C

Join

Join Example:

```
SELECT s.student_id, s.name, b.book_id, b.price
FROM students s, books b
WHERE s.student_id = b.student_id
AND b.price > 90 ;
```

Students table:

STUDENT_ID	NAME	YEAR
1	STUDENT_1	I
2	STUDENT_2	II
3	STUDENT_3	III
4	STUDENT_4	IV

Books table:

BOOK_ID	STUDENT_ID	PRICE
1	1	40
2	2	50
3	3	70
4	1	100
5	2	120
6	4	90
7	3	200
8	2	150

Join Result:

STUDENT_ID	NAME	BOOK_ID	PRICE
1	STUDENT_1	4	100
2	STUDENT_2	5	120
3	STUDENT_3	7	200
2	STUDENT_2	8	150

Inner Join

Inner Join Example:

```
SELECT s.student_id, s.name, SUM(b.price)
FROM students s INNER JOIN books b
ON s.student_id = b.student_id
GROUP BY b.price ;
```

Students table:

STUDENT_ID	NAME	YEAR
1	STUDENT_1	I
2	STUDENT_2	II
3	STUDENT_3	III
4	STUDENT_4	IV

Books table:

BOOK_ID	STUDENT_ID	PRICE
1	1	40
2	2	50
3	3	70
4	1	100
5	2	120
6	4	90
7	3	200
8	2	150

Inner Join Result:

STUDENT_ID	NAME	PRICE
1	STUDENT_1	140
2	STUDENT_2	320
3	STUDENT_3	270
4	STUDENT_4	90

Left Join

Left Join Example:

```
SELECT s.student_id, s.name, b.price
FROM students s LEFT JOIN books b
ON s.student_id = b.student_id
ORDER BY s.student_id ;
```

Students table:

STUDENT_ID	NAME	YEAR
1	STUDENT_1	I
2	STUDENT_2	II
3	STUDENT_3	III
4	STUDENT_4	IV
5	STUDENT_5	I
6	STUDENT_6	IV

Books table:

BOOK_ID	STUDENT_ID	PRICE
1	1	40
2	2	50
3	3	70
4	1	100
5	2	120
6	4	90
7	3	200
8	2	150

Left Join Result:

STUDENT_ID	NAME	PRICE
1	STUDENT_1	40
1	STUDENT_1	100
2	STUDENT_2	50
2	STUDENT_2	120
2	STUDENT_2	150
3	STUDENT_3	70
3	STUDENT_3	200
4	STUDENT_4	90
5	STUDENT_5	
6	STUDENT_6	

Right Join

Right Join Example:

```
SELECT * FROM employee e RIGHT JOIN department d
ON e.DEP_ID = d.DEP_ID
ORDER BY d.DEP_ID ;
```

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	EMPLOYEE_1	21
2	EMPLOYEE_2	22
3	EMPLOYEE_3	23
4	EMPLOYEE_4	24

5	EMPLOYEE_5

Department table:

DEP_ID	DEP_NAME
21	DEP_21
22	DEP_22
23	DEP_23
24	DEP_24
25	DEP_25

Right Join Result:

EMPLOYEE_ID	NAME	DEP_ID	DEP_ID	DEP_NAME
1	EMPLOYEE_1	21	21	DEP_21
2	EMPLOYEE_2	22	22	DEP_22
3	EMPLOYEE_3	23	23	DEP_23
4	EMPLOYEE_4	24	24	DEP_24
			25	DEP_25

Full Join

Full Join Example:

```
SELECT * FROM employee e FULL JOIN department d
ON e.DEP_ID = d.DEP_ID
ORDER BY e.employee_id ;
```

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	EMPLOYEE_1	21
2	EMPLOYEE_2	22
3	EMPLOYEE_3	23
4	EMPLOYEE_4	24
5	EMPLOYEE_5	

Department table:

DEP_ID	DEP_NAME
21	DEP_21
22	DEP_22
23	DEP_23
24	DEP_24
25	DEP_25

Full Join Result:

EMPLOYEE_ID	NAME	DEP_ID	DEP_ID	DEP_NAME
1	EMPLOYEE_1	21	21	DEP_21
2	EMPLOYEE_2	22	22	DEP_22
3	EMPLOYEE_3	23	23	DEP_23
4	EMPLOYEE_4	24	24	DEP_24
5	EMPLOYEE_5			
			25	DEP_25

Union

Union Syntax:

```
SELECT column_name(s) FROM table_name_a
UNION
SELECT column_name(s) FROM table_name_b
```

Union All Syntax:

```
SELECT column_name(s) FROM table_name_a
UNION ALL
SELECT column_name(s) FROM table_name_b
```

Employee_a		Employee_b	
id	**name**	**id**	**name**
1	Martin	1	David
2	Carol	2	Barbara
3	Davis	3	Carol
4	Sandra	4	Sandra

UNION Example:

SELECT * FROM employee_a UNION SELECT * FROM employee_b;

UNION Result:

1 Martin
2 Carol
3 Davis
4 Sandra
1 David
2 Barbara
3 Carol

UNION ALL Example:

SELECT * FROM employee_a UNION ALL SELECT * FROM employee_b;

UNION ALL Result:

1 Martin
2 Carol
3 Davis
4 Sandra
1 David
2 Barbara
3 Carol
4 Sandra

TOP

TOP Syntax:

SELECT **TOP number** column_name(s) FROM table_name
SELECT **TOP percent** column_name(s) FROM table_name

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	EMPLOYEE_1	21

2	EMPLOYEE_2	22
3	EMPLOYEE_3	23
4	EMPLOYEE_4	24

TOP number Example:

SELECT TOP 3 * FROM employee;

TOP Result:

EMPLOYEE_ID	NAME	DEP_ID
1	EMPLOYEE_1	21
2	EMPLOYEE_2	22
3	EMPLOYEE_3	23

TOP percent Example:

SELECT TOP 50 PERCENT * FROM employee;

TOP Result:

EMPLOYEE_ID	NAME	DEP_ID
1	EMPLOYEE_1	21
2	EMPLOYEE_2	22

Wildcard

Wildcard	Definition
%	Represents zero or more characters
_	Represents exactly one character
[char list]	Represents any single character in charlist
[^char list] or [!char list]	Represents any single character not in charlist

Students table:

ID	NAME	STATE

1	Tom	Arizona
2	Martin	Texas
3	Helen	Florida
4	Tania	California
5	Harry	Colorado

_ Wildcard Example:

Select the student with a name that starts with any character, followed by "ar".
SELECT * FROM students WHERE name LIKE '_ar';

_ Wildcard Result:

ID	NAME	STATE
2	Martin	Texas
5	Harry	Colorado

[char list] Wildcard Example:

Select the student with a name that starts with any character from char list.
SELECT * FROM students WHERE name LIKE '[tma]%';

[char list] Wildcard Result:

1	Tom	Arizona
2	Martin	Texas
4	Tania	California

[!char list] Wildcard Example:

Select the student with a name that do not starts with any character from char list.
SELECT * FROM students WHERE name LIKE '[!tma]%';

[!char list] Wildcard Result:

3	Helen	Florida
5	Harry	Colorado

In

In Syntax:

SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,value3,...)

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	John	33
2	Samantha	34
3	Bill	35
4	James	36
5	Sandra	37

In Example:

SELECT * FROM employee WHERE name IN ('Samantha', 'Bill', 'Sandra');

In Result:

2	Samantha	34
3	Bill	35
5	Sandra	37

Between

Between Syntax:

SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value_a AND value_b

Employee table:

EMPLOYEE_ID	NAME	DEP_ID
1	John	33
2	Samantha	34
3	Bill	35
4	James	36
5	Sandra	37

Between Example:

```
SELECT * FROM employee WHERE dep_id BETWEEN 34 AND 36;
```

Between Result:

2	Samantha	34
3	Bill	35
4	James	36

ISNULL

ISNULL Syntax:

```
SELECT ISNULL(column_name,0) FROM table_name
```

Sales table:

ID	PRICE	NAME
1	100	A
2		B
3	600	C
4		D

ISNULL Example:

```
SELECT id, ISNULL(price,0), name FROM store;
```

ISNULL Result:

ID	PRICE	NAME
1	100	A
2	0	B
3	600	C
4	0	D

Create Table

Create Database Syntax:

CREATE DATABASE database_name

Create Database Example:

CREATE DATABASE new_dba;

Create Table

Create Table Syntax:

```
CREATE TABLE new_table
(
column_name_1 datatype,
column_name_2 datatype,
....
column_name_n datatype
);
```

Create Table Example:

```
CREATE TABLE sales
(
id int,
price int,
name varchar(50)
);
```

Create Index

Create Index Syntax:

```
CREATE INDEX my_index
ON my_table (column_name)
```

Create Unique Index Syntax:

```
CREATE UNIQUE INDEX my_index
ON my_table (column_name)
```

Create View

Create View Syntax:

```
CREATE VIEW my_view_name AS
SELECT column_name(s)
FROM my_table_name
WHERE condition
```

Create View Example:

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

```
CREATE VIEW sales_view AS
SELECT id, price, name
FROM sales
WHERE price=500;
```

Create View Result:

ID	PRICE	NAME
2	500	B
4	500	D

Increment

Identity Syntax:

```
CREATE TABLE new_table
(
column_name_1 PRIMARY KEY IDENTITY,
column_name_2 datatype,
....
column_name_n datatype
);
```

Identity Example:

```
CREATE TABLE sales
(
id int PRIMARY KEY IDENTITY,
price int,
name varchar(50)
);
```

Drop

Drop Table Syntax:

```
DROP TABLE table_name;
```

Drop Database Syntax:

```
DROP DATABASE database_name;
```

Drop Index Syntax:

DROP INDEX table_name.index_name;

Truncate Table Syntax:

TRUNCATE TABLE table_name;

Alter Table

Alter Table
Add Column
Alter Column
Rename Column
Drop Column

Add Column

Add Column Syntax:

ALTER TABLE table_name
ADD column_name data_type

Employee table:

Column name	Data_type
id	int
name	varchar(250)

Add Column Example:

ALTER TABLE employee ADD (dep_id int, address varchar(100));

Add Column Result:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
address	varchar(100)

Alter Column

Alter Column Syntax:

ALTER TABLE table_name
ALTER COLUMN column_name data_type

Employee table:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
address	varchar(100)

Alter Column Example:

ALTER TABLE employee ALTER COLUMN address varchar(400);

Alter Column Result:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
address	varchar(400)

Rename Column

Rename Column Syntax:

EXEC sp_rename 'Table.Old_Column', 'New_Column', 'COLUMN'

Employee table:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
address	varchar(400)

Rename Column Example:

EXEC sp_rename 'employee.address', 'new_address', 'COLUMN' ;

Rename Column Result:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
new_address	varchar(400)

Drop Column

Drop Column Syntax:

ALTER TABLE table_name
DROP COLUMN column_name

Employee table:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int
address	varchar(400)

Drop Column Example:

ALTER TABLE employee DROP COLUMN address;

Drop Column Result:

Column name	Data_type
id	int
name	varchar(250)
dep_id	int

AVG

AVG Syntax:

SELECT AVG(column_name) FROM table_name

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

AVG Example:

SELECT AVG(price) FROM store;

AVG Result:

525

Count

Count Syntax:

SELECT COUNT(column_name) FROM table_name
SELECT COUNT(*) FROM table_name

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Count Example 1:

SELECT COUNT(id) FROM store WHERE price=500;

Count Result: 2

Count Example 2:

SELECT COUNT(*) FROM store;

Count Result: 4

Max

Max Syntax:

SELECT MAX(column_name) FROM table_name

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Max Example:

SELECT MAX(price) FROM store;

Max Result: 900

Min

Min Syntax:

SELECT MIN(column_name) FROM table_name

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Min Example:

SELECT MIN(price) FROM store;

Min Result: 200

Sum

Sum Syntax:

SELECT SUM(column_name) FROM table_name

Sales table:

ID	PRICE	NAME
1	200	A
2	500	B
3	900	C
4	500	D

Sum Example:

SELECT SUM(price) FROM store;

Sum Result: 2100

Mid

Mid Syntax:

SELECT MID(column_name,start[,length]) FROM table_name

Students table:

ID	NAME	State
1	Tom	Arizona
2	Linda	Texas
3	Helen	Florida
4	Robert	California

Mid Example:

SELECT state, MID(state,1,4) FROM students;

Mid Result:

State	MID(state,1,3)
Arizona	Ari
Texas	Tex
Florida	Flo
California	Cal

Len

Len Syntax:

SELECT LEN(column_name) FROM table_name

Students table:

ID	NAME	State
1	Tom	Arizona
2	Linda	Texas
3	Helen	Florida
4	Robert	California

Len Example:

SELECT state, LEN(state) FROM students;

Len Result:

State	LEN(state)
Arizona	7
Texas	5
Florida	7
California	10

Round

Round Syntax:

SELECT ROUND(column_name,decimal precision) FROM table_name

Sales table:

ID	PRICE	NAME
1	25.845	A
2	26.97	B

3	27.9	C
4	28.34	D

Round Example 1:

SELECT id, ROUND(price,1) FROM store;

Round Result:

ID	PRICE
1	25.8
2	26.9
3	27.9
4	28.3

Round Example 2:

SELECT id, ROUND(price,0) FROM store;

Round Result:

ID	PRICE
1	26
2	27
3	28
4	28

Now

Now Syntax:

SELECT NOW() FROM my_table

Sales table:

ID	PRICE	NAME
1	25.845	A
2	26.97	B

3	27.9	C
4	28.34	D

Now Example:

SELECT id, price, NOW() as PriceDate FROM store;

Now Result:

Id	Price	PriceDate
1	25.845	12/9/2012 15:30:23 PM
2	26.97	12/9/2012 15:30:23 PM
3	27.9	12/9/2012 15:30:23 PM
4	28.34	12/9/2012 15:30:23 PM

UCase

UCase Syntax:

SELECT UCASE(column_name) FROM table_name

Students table:

ID	NAME	State
1	Tom	Arizona
2	Linda	Texas
3	Helen	Florida
4	Robert	California

UCase Example:

SELECT name, UCASE(name) FROM students;

UCase Result:

Name	UCASE(name)
Tom	TOM
Linda	LINDA

Helen	HELEN
Robert	ROBERT

LCase

LCase Syntax:

SELECT LCASE(column_name) FROM table_name

Students table:

ID	NAME	State
1	Tom	Arizona
2	Linda	Texas
3	Helen	Florida
4	Robert	California

LCase Example:

SELECT name, LCASE(name) FROM students;

LCase Result:

Name	LCASE(name)
Tom	tom
Linda	linda
Helen	helen
Robert	robert

Stored Procedures

Stored Procedures

Create Procedure
Create Function
Call Stored Procedure
Drop Stored Procedure
Rename Stored Procedure

Create Stored Procedure

Create Procedure Example:

Customers Table

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_TYPE
1	CUSTOMER_1	CC
2	CUSTOMER_2	I
3	CUSTOMER_3	SM
4	CUSTOMER_4	CC

Contracts Table

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
CREATE PROCEDURE SalesByCustomer
@CustomerName nvarchar(50)
AS
SELECT c.customer_name, sum(ctr.amount) AS TotalAmount
FROM customers c, contracts ctr
WHERE c.customer_id = ctr.customer_id
AND c.customer_name = @CustomerName
GROUP BY c.customer_name
ORDER BY c.customer_name
GO

EXEC SalesByCustomer 'CUSTOMER_1'
GO
```

Customer_Name	TotalAmount
CUSTOMER_1	1400

Create Function

1. Create Function Example

```
CREATE FUNCTION CtrAmount ( @Ctr_Id int(10) )
RETURNS MONEY
AS
BEGIN
DECLARE @CtrPrice MONEY
SELECT @CtrPrice = SUM(amount)
FROM Contracts
WHERE contract_id = @Ctr_Id
RETURN(@CtrPrice)
END
GO

SELECT * FROM CtrAmount(345)
GO
```

2. Create Function Example

```
CREATE FUNCTION function_name (@PRODUCT_ID Int)
RETURNS @ProductsList Table
(Product_Id Int,
Product_Dsp nvarchar(150),
Product_Price Money )
AS
BEGIN
IF @PRODUCT_ID IS NULL
BEGIN
INSERT INTO @ProductsList (Product_Id, Product_Dsp, Product_Price)
SELECT Product_Id, Product_Dsp, Product_Price
FROM Products
END
ELSE
BEGIN
INSERT INTO @ProductsList (Product_Id, Product_Dsp, Product_Price)
SELECT Product_Id, Product_Dsp, Product_Price
FROM Products
WHERE Product_Id = @PRODUCT_ID
END
RETURN
END
GO
```

Call Stored Procedure

Call Stored Procedure Example:

```
EXEC SalesByCustomer 'CUSTOMER_1'
GO
```

Execute procedure in 20 minutes

```
BEGIN
WAITFOR DELAY "0:20:00"
EXEC SalesByCustomer 'CUSTOMER_1'
END
```

Drop Stored Procedure

Drop Stored Procedure Syntax:

DROP PROCEDURE stored_procedure_name

Drop Stored Procedure Example:

DROP PROCEDURE SalesByCustomer
GO

Rename Stored Procedure

Rename Stored Procedure Syntax:

sp_rename 'old_procedure_name', 'new_procedure_name'

Rename Stored Procedure Example:

EXEC sp_rename 'SalesByCustomer', 'NewSalesByCustomer';
GO

System Stored Procedures

The Transact SQL language allow you to use various system stored procedures like: sp_tables, sp_table_privileges, sp_stored_procedures, sp_cursor, sp_executesql, sp_rename, sp_lock, sp_execute, sp_help.

Sp_addextendedproperty

The sp_addextendedproperty is part of Database Engine Stored Procedures and adds a new extended property to a database object.

Sp_addextendedproperty Syntax

```
sp_addextendedproperty
[ @name = 'property_name' ]
[ , [ @value = 'value' ]
  [ , [ @level0type = 'level0_object_type' ]
    , [ @level0name = 'level0_object_name' ]
      [ , [ @level1type = 'level1_object_type' ]
        , [ @level1name = 'level1_object_name' ]
          [ , [ @level2type = 'level2_object_type' ]
            , [ @level2name = 'level2_object_name' ]
] ] ] ] ;
```

Add an extended property to a table:

```
USE model;
GO
EXEC sys.sp_addextendedproperty
@name = N'Test example',
@value = N'This is an example.',
@level0type = N'SCHEMA', @level0name = 'dbo',
@level1type = N'TABLE', @level1name = 'my_table';
GO
```

Sp_columns

The sp_columns is part of Catalog Stored Procedures and return column information for the specified objects that can be queried in the database.

Sp_columns syntax:

```
sp_columns [ @table_name = 'Object name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ,
[ @column_name = 'Column name.' ] ,
[ @ODBCVer = 'ODBC Version 2 or 3.' ] ;
```

Sp_columns example:

```
USE model;
GO
EXEC sp_columns
@table_name = 'students',
@table_owner = 'dbo';
```

Sp_column_privileges

The sp_column_privileges is part of Catalog Stored Procedures and return column privilege information for a single table in the current database.

Sp_column_privileges syntax:

```
sp_column_privileges [ @table_name = 'Table name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ,
[ @column_name = 'Column name.' ] ;
```

Sp_column_privileges example:

```
USE model;
GO
EXEC sp_column_privileges
@table_name = 'students',
@table_owner = 'dbo';
```

Sp_special_columns

The sp_special_columns is part of Catalog Stored Procedures and return the optimal set of columns that uniquely identify a row in the table.

Sp_special_columns syntax:

```
sp_special_columns [ @table_name = 'Table name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ,
[ @col_type = 'Column type.' ] ,
[ @scope = 'The minimum required scope of the ROWID.' ] ,
[ @nullable = 'The special columns can accept a null value.' ] ,
[ @ODBCVer = 'ODBC Version 2 or 3.' ] ;
```

Sp_special_columns example:

```
USE model;
GO
EXEC sp_special_columns
@table_name = 'students',
@table_owner = 'dbo';
```

Sp_configure

The sp_configure is part of Database Engine Stored Procedures and displays or changes global configuration settings for the current server.

Sp_configure syntax:

```
sp_configure [ [ @configname = 'option_name' ] , [ @configvalue = ] 'value' ];
```

Sp_configure example:

```
USE model;
GO
EXEC sp_configure 'show advanced option', '1';
```

Sp_databases

The sp_databases is part of Catalog Stored Procedures and return a list of databases from an instance of the SQL Server or are accessible through a database gateway.

Sp_databases syntax:

```
sp_databases ;
```

Sp_databases example:

```
USE model;
GO
EXEC sp_databases ;
```

Sp_execute

The sp_execute is part of Database Engine Stored Procedures and executes a prepared Transact-SQL statement using a specified handle and optional parameter value.

Sp_execute syntax:

```
sp_execute handle OUTPUT, [bound_param ] [,...n ];
```

Sp_execute example:

```
USE model;
GO
EXEC sp_execute 6, null;
GO
```

Sp_executesql

The sp_executesql is part of Database Engine Stored Procedures and executes a Transact-SQL statement or batch that can be reused many times.

Sp_executesql syntax:

```
sp_executesql [ @stmt = ] statement ;
```

Sp_executesql example:

```
USE model;
GO
EXECUTE sp_executesql
N'SELECT * FROM students WHERE id= @id',
N'@id int',
@id = 3;
```

Sp_fkeys

The sp_fkeys is part of Catalog Stored Procedures and return logical foreign key information for the current database.

Sp_fkeys syntax:

```
sp_fkeys [ @pktable_name = 'Table name.' ] ,
[ @pktable_owner = 'The database user who created the table.' ] ,
[ @pktable_qualifier = 'Database name.' ] ,
[ @fktable_name = 'Table name.' ] ,
[ @fktable_owner = 'The database user who created the table.' ] ,
[ @fktable_qualifier = 'Database name.' ] ;
```

Sp_fkeys example:

```
USE model;
GO
EXEC sp_fkeys
@pktable_name = 'students',
@pktable_owner = 'dbo',
@pktable_qualifier = 'model',
@fktable_name = 'students',
@fktable_owner = 'dbo',
@fktable_qualifier = 'model';
```

Sp_help

The sp_help is part of Database Engine Stored Procedures and reports information about a database object or a data type.

Sp_help syntax:

```
sp_help [ @objname = 'Object name.' ] ;
```

Sp_help example:

```
USE model;
GO
EXEC sp_help;
GO
```

```
USE model;
GO
EXEC sp_help 'students';
GO
```

Sp_helpdb

The sp_helpdb is part of Database Engine Stored Procedures and shows information about a specified database or all databases.

Sp_helpdb syntax:

```
sp_helpdb [ @dbname= 'Database name.' ] ;
```

Sp_helpdb example:

```
EXEC sp_helpdb;
GO

EXEC sp_helpdb N'model';
GO
```

Sp_helpindex

The sp_helpindex is part of Database Engine Stored Procedures and reports information about the indexes on a table or view.

Sp_helpindex syntax:

```
sp_helpindex [ @objname = 'Object name.' ] ;
```

Sp_helpindex example:

```
USE model;
GO
EXEC sp_helpindex N'Departments';
GO
```

Sp_lock

The sp_lock is part of Database Engine Stored Procedures and shows information about locks.

Sp_lock syntax:

sp_lock [[@spid =] 'session ID'] ;

Sp_lock example:

```
USE model;
GO
EXEC sp_lock;
GO
```

Sp_monitor

The sp_monitor is part of Database Engine Stored Procedures and shows statistics about Microsoft SQL Server.

Sp_monitor syntax:

sp_monitor ;

Sp_monitor example:

```
USE model;
GO
EXEC sp_monitor;
GO
```

Sp_prepare

The sp_prepare is part of Database Engine Stored Procedures and prepares a parameterized statement and returns a statement handle for execution.

Sp_prepare syntax:

sp_prepare handle OUTPUT, params, statement, options ;

Sp_prepare example:

```
USE model;
GO
DECLARE @Param1 int;
EXEC sp_prepare @Param1 output,
N'@Param1 nvarchar(250), @Param2 nvarchar(250)',
N'SELECT database_id, name FROM sys.databases
```

```
WHERE name=@Param1 AND state_desc = @Param2';
EXEC sp_execute @Param1, N'Test', N'ONLINE';
GO
```

Sp_pkeys

The sp_pkeys is part of Catalog Stored Procedures and return primary key information for a single table in the current database.

Sp_pkeys syntax:

```
sp_pkeys [ @table_name = 'Table name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ;
```

Sp_pkeys example:

```
USE model;
GO
EXEC sp_pkeys
@table_name = 'students',
@table_owner = 'dbo',
@table_qualifier = 'model';
GO
```

Sp_rename

The sp_rename is part of Database Engine Stored Procedures and rename the name of an object (table, index, column or alias data type) in the current database.

Sp_rename syntax:

```
sp_rename
[ @objname = 'Object name' ] ,
[ @newname = 'New object name' ] [ , [ @objtype = 'Object type' ] ] ;
```

Rename a table:

```
USE model;
GO
EXEC sp_rename 'Departments', 'test';
GO
```

Rename a column:

```
USE model;
GO EXEC sp_rename 'my_table.b', 'c', 'COLUMN';
GO
```

Sp_renamedb

The sp_renamedb is part of Database Engine Stored Procedures and changes the name of a database.

Sp_renamedb syntax:

```
sp_renamedb
[ @dbname = 'Old database name' ] ,
[ @newname = 'New database name' ] ;
```

Rename a database name:

```
USE master;
GO
CREATE DATABASE myTest;
GO
EXEC sp_renamedb N'myTest', N'Test';
GO
```

Sp_tables

The sp_tables is part of Catalog Stored Procedures and return a list of objects (table or view) that can be queried in the current environment.

Sp_tables syntax:

```
sp_tables [ @table_name = 'Table name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ,
[ @table_type = "Table, system table, or view." ] ;
```

Sp_tables example:

```
USE model;
GO
EXEC sp_tables ;
GO
```

Sp_table_privileges

The sp_table_privileges is part of Catalog Stored Procedures and return a list of table permissions for the specified table or tables.

Sp_table_privileges syntax:

```
sp_table_privileges [ @table_name = 'Table name.' ] ,
[ @table_owner = 'The database user who created the table.' ] ,
[ @table_qualifier = 'Database name.' ] ;
```

Sp_table_privileges example:

```
USE model;
GO
EXEC sp_table_privileges
@table_name = '%';
```

Sp_server_info

The sp_server_info is part of Catalog Stored Procedures and return a list of attribute names and matching values for SQL Server.

Sp_server_info syntax:

```
sp_server_info [@attribute_id = 'attribute_id'];
```

Sp_server_info example:

```
USE model;
GO
EXEC sp_server_info
@attribute_id = '1';
```

Sp_stored_procedures

The sp_table_privileges is part of Catalog Stored Procedures and return a list of stored procedures in the current environment.

Sp_stored_procedures syntax:

```
sp_stored_procedures [ @sp_name = 'Procedure name.' ] ,
[ @sp_owner = 'The schema or database user who created.' ] ,
```

[@sp_qualifier = 'Database name.'] ;

Sp_stored_procedures example:

```
USE model;
GO EXEC sp_stored_procedures ;
GO
```

Sp_unprepare

The sp_unprepare is part of Database Engine Stored Procedures and discards the execution statement created by the sp_prepare stored procedure.

Sp_unprepare syntax:

```
sp_unprepare handle ;
```

Sp_unprepare example:

```
USE model;
GO
DECLARE @Param1 int;
EXEC sp_prepare @Param1 output,
N'@Param1 nvarchar(100), @Param2 nvarchar(100)',
N'SELECT database_id, name FROM sys.databases
WHERE name=@Param1 AND state_desc = @Param2';
EXEC sp_execute @Param1, N'myDatabase', N'ONLINE';
EXEC sp_unprepare @Param1;
GO
```

Sp_updatestats

The sp_updatestats is part of Database Engine Stored Procedures and runs update statistics against all tables in the current database.

Sp_updatestats syntax:

```
sp_updatestats [ @resample = 'resample'];
```

Sp_updatestats example:

```
USE model;
GO
```

```
EXEC sp_updatestats;
GO
```

Sp_who

The sp_who is part of Database Engine Stored Procedures and shows information about current users, sessions and processes.
Sp_who syntax:

```
sp_who [ [ @loginame = ] 'login' | session ID | 'ACTIVE' ];
```

List all processes:

```
USE model;
GO
EXEC sp_who;
GO
```

List active processes:

```
USE model;
GO
EXEC sp_who 'active';
GO
```

Table Joins

Table joins

Inner Join
Left Join
Right Join
Self Join

Table Inner Join

Inner Join Example:

Customers Table

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_TYPE
1	CUSTOMER_1	CC
2	CUSTOMER_2	I
3	CUSTOMER_3	SM
4	CUSTOMER_4	CC

Contracts Table

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
SELECT c.customer_id, c.customer_name, ctr.contract_id, ctr.amount
FROM customers c, contracts ctr
WHERE c.customer_id = ctr.customer_id
AND ctr.amount > 900 ;
```

Result:

Customer_Id	Customer_Name	Contract_Id	Amount
1	CUSTOMER_1	4	1000
2	CUSTOMER_2	8	1500
2	CUSTOMER_2	5	1200
3	CUSTOMER_3	7	2000

Table Left Join

Left Join Example:

Customers Table

CUSTOMER_ID	CUSTOMER_NAME	CONTACT_NAME
1	CUSTOMER_1	NAME_1
2	CUSTOMER_2	NAME_2
3	CUSTOMER_3	NAME_3
4	CUSTOMER_4	NAME_4
5	CUSTOMER_5	
6	CUSTOMER_6	

Contracts Table

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
SELECT c.customer_id, c.customer_name, ctr.contract_id, ctr.amount
FROM customers c LEFT JOIN contracts ctr
ON c.customer_id = ctr.customer_id
ORDER BY c.customer_id ;
```

Result:

Customer_Id	Customer_Name	Contract_Id	Amount
1	CUSTOMER_1	1	400
1	CUSTOMER_1	4	1000
2	CUSTOMER_2	2	500
2	CUSTOMER_2	5	1200
2	CUSTOMER_2	8	1500
3	CUSTOMER_3	3	700
3	CUSTOMER_3	7	2000
4	CUSTOMER_4	6	900
5	CUSTOMER_5		
6	CUSTOMER_6		

Table Right Join

Right Join Example:

Customers Table

CUSTOMER_ID	CUSTOMER_NAME	USER_NAME
1	CUSTOMER_1	SYSTEM
2	CUSTOMER_2	ELLEN
3	CUSTOMER_3	TOM
4	CUSTOMER_4	STEVE
5	CUSTOMER_5	
6	CUSTOMER_6	

Users Table

USER_NAME	DEPARTMENT_NAME
SYSTEM	IT
TOM	SALES
STEVE	SALES
ELLEN	SALES
JOHN	IT

```
SELECT c.customer_id, c.customer_name, c.user_name
FROM users u RIGHT JOIN customers c
ON c.user_name = u.user_name
ORDER BY c.customer_id ;
```

Result:

Customer_Id	Customer_Name	User_Name
1	CUSTOMER_1	SYSTEM
2	CUSTOMER_2	ELLEN
3	CUSTOMER_3	TOM
4	CUSTOMER_4	STEVE
5	CUSTOMER_5	
6	CUSTOMER_6	

Table Self Join

Self Join Example:

Users Table

ID	USER_NAME	DEPARTMENT_NAME	MANAGER_ID
1	SYSTEM	IT	1
2	TOM	SALES	2
3	STEVE	SALES	2
4	ELLEN	SALES	2
5	JOHN	IT	1

```
SELECT u.id, u.user_name, m.id manager_id, m.user_name manager
 FROM users u, users m
WHERE u.manager_id = m.id
ORDER BY u.id ;
```

Result:

Id	User_Name	Manager_Id	Manager
1	SYSTEM	1	SYSTEM
2	TOM	2	TOM
3	STEVE	2	TOM
4	ELLEN	2	TOM
5	JOHN	1	SYSTEM

Transactions

A transaction is a single logical unit of work and it is composed of several sql server statements. The transaction begins with the first sql server statement executed and ends when the transaction is saved or rolled back.

T-SQL Transaction Statements

- BEGIN DISTRIBUTED TRANSACTION
- BEGIN TRANSACTION
- COMMIT TRANSACTION
- COMMIT WORK
- ROLLBACK TRANSACTION
- ROLLBACK WORK
- SAVE TRANSACTION

Begin Distributed Transaction

A distributed transaction is an operations transaction that can be run on multiple different servers.

Begin distributed transaction syntax:

```
BEGIN DISTRIBUTED { TRAN | TRANSACTION }
[ transaction_name | @transaction_name_variable ] ;
```

Begin distributed transaction example:

```
USE model;
GO
BEGIN DISTRIBUTED TRANSACTION;
--> Delete students from local instance.
DELETE students WHERE id = 7;
--> Delete students from remote instance.
DELETE RemoteServer.students WHERE id = 7;
COMMIT TRANSACTION;
GO
```

Begin Transaction

The Begin Transaction is the start point of an explicit transaction and increments @@TRANCOUNT by 1.

Begin transaction syntax:

```
BEGIN { TRAN | TRANSACTION }
[ { transaction_name | @transaction_name_variable }
[ WITH MARK [ 'transaction description' ] ] ];
```

Begin transaction example:

```
BEGIN TRANSACTION DelStudent WITH MARK N'Delete student';
GO
USE model;
GO
DELETE FROM students WHERE id = 7 and section = 'History';
GO
COMMIT TRANSACTION DelStudent;
GO
```

COMMIT Transaction

The COMMIT Transaction is the end point of a successful implicit or explicit transaction.

When @@TRANCOUNT is 1, all data modifications performed are commit and the resources held by the transaction are released, and decrements @@TRANCOUNT to 0. When @@TRANCOUNT is greater than 1, COMMIT TRANSACTION decrements @@TRANCOUNT only by 1 and the transaction stays active.

Commit transaction syntax:

```
COMMIT [ { TRAN | TRANSACTION } [ transaction_name |
@transaction_name_variable ] ]
[ WITH ( DELAYED_DURABILITY = { OFF | ON } ) ] ;
```

Commit transaction example:

```
USE model;
GO
BEGIN TRANSACTION;
GO
DELETE FROM students WHERE id = 7 and section = 'History';
GO
insert into students(id,first_name, last_name, gender,city, country, section)
values(7,'Ashley','THOMPSON','F','London','Liverpool', 'History');
GO
COMMIT TRANSACTION;
GO
```

COMMIT WORK

The COMMIT WORK transaction is the end point of a successful transaction.

Commit work syntax:

```
COMMIT [ WORK ] ;
```

Commit work example:

```
USE model;
GO
BEGIN TRANSACTION;
GO
DELETE FROM students WHERE id = 7 ;
GO
insert into students(id,first_name, last_name, gender,city, country, section)
values(7,'Ashley','THOMPSON','F','Liverpool','England', 'History');
GO
COMMIT WORK;
GO
```

ROLLBACK Transaction

The ROLLBACK Transaction is an operation that rolls back an unsuccessful explicit or implicit transaction to the beginning of the transaction or to a save point inside the transaction.

Rollback transaction syntax:

```
ROLLBACK { TRAN | TRANSACTION }
[ transaction_name | @transaction_name_variable
| savepoint_name | @savepoint_variable ] ;
```

Rollback transaction example:

```
USE model;
GO
BEGIN TRANSACTION;
GO
insert into students(id,first_name, last_name, gender,city, country, section)
values(8,'Alysia','MARTIN','F','Toronto','Canada', 'Biology');
GO
ROLLBACK TRANSACTION;
GO
```

ROLLBACK WORK

The ROLLBACK WORK is an transaction operation that rolls back a user specified transaction.

Rollback work syntax:

```
ROLLBACK [ WORK ] ;
```

Rollback work example:

```
USE model;
GO
BEGIN TRANSACTION;
GO
```

```
insert into students(id,first_name, last_name, gender,city, country, section)
values(8,'Alysia','MARTIN','F','Toronto','Canada', 'Biology');
GO
ROLLBACK WORK;
GO
```

SAVE Transaction

The SAVE transaction sets a save point within a transaction.

Save transaction syntax:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable } ;
```

Save transaction example:

```
USE model;
GO
DECLARE @Counter INT;
SET @Counter = @@TRANCOUNT;
IF @Counter > 0
SAVE TRANSACTION my_savepoint;
ELSE
BEGIN TRANSACTION;
GO
insert into students(id,first_name, last_name, gender,city, country, section)
values(8,'Alysia','MARTIN','F','Toronto','Canada', 'Biology');
GO
COMMIT TRANSACTION;
GO
```

Triggers

In this chapter you can learn how to work with triggers using operations like create, alter, rename, drop, enable, disable.

Trigger operations

- Create Trigger
- Alter Trigger
- Rename Trigger
- Drop Trigger
- Enable Trigger
- Disable Trigger

Create Trigger

Create Trigger Syntax

DML Trigger

```
CREATE TRIGGER trig_name
ON { table_name | view_name }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement [ ; ] }
GO
```

DDL Trigger

```
CREATE TRIGGER trig_name
ON { ALL SERVER | DATABASE }
{ FOR | AFTER }
{ CREATE, ALTER, DROP, GRANT, DENY, REVOKE, UPDATE STATISTICS }
AS { sql_statement [ ; ] }
GO
```

Logon Trigger

```
CREATE TRIGGER trig_name
ON ALL SERVER
{ FOR| AFTER }
LOGON
AS { sql_statement [ ; ] }
GO
```

Alter Trigger

Alter Trigger Syntax

DML Trigger

```
ALTER TRIGGER trig_name
ON { table_name | view_name }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement [ ; ] }
GO
```

DDL Trigger

```
ALTER TRIGGER trig_name
ON { ALL SERVER | DATABASE }
{ FOR | AFTER }
{ CREATE, ALTER, DROP, GRANT, DENY, REVOKE, UPDATE STATISTICS }
AS { sql_statement [ ; ] }
GO
```

Logon Trigger

```
ALTER TRIGGER trig_name
ON ALL SERVER
{ FOR| AFTER }
LOGON
AS { sql_statement [ ; ] }
GO
```

Rename Trigger

Rename Trigger Syntax

sp_rename 'Old_Trigger_Name', 'New_Trigger_Name';

Rename Trigger Example

```
CREATE TRIGGER Customers_Op
ON Customers
AFTER UPDATE
AS
UPDATE Customers
SET address = UPPER('ADDRESS_5')
WHERE customer_id = 5
GO
```

sp_rename 'Customers_Op', 'NewCustomers_Op';

Drop Trigger

Drop Trigger Syntax

DROP TRIGGER trigger_name;

Drop Trigger Example

```
CREATE TRIGGER Customers_Op
ON Customers
AFTER UPDATE
AS
UPDATE Customers
SET address = UPPER('ADDRESS_5')
WHERE customer_id = 5
```

GO

DROP TRIGGER Customers_Op;

Enable Trigger

Enable Trigger Syntax

ENABLE TRIGGER { [schema_name .] trigger_name [,...n_trigger_name] | ALL }
ON { table_name | DATABASE | ALL SERVER } [;]

Enable Trigger Example 1

ALTER TABLE Customers ENABLE TRIGGER ALL;

Enable Trigger Example 2

ENABLE Trigger master.Customers_Op ON master.Customers ;

Disable Trigger

Disable Trigger Syntax

ALTER TABLE table_name DISABLE TRIGGER ALL;
DISABLE Trigger master.trigger_name ON master.table_name ;
DISABLE Trigger ALL ON ALL SERVER;

Disable Trigger Example

ALTER TABLE Customers DISABLE TRIGGER ALL;
DISABLE Trigger system.Customers_Op ON system.Customers ;
DISABLE Trigger ALL ON ALL SERVER;

Views

Create View

Create View Example

Customers table

CUSTOMER_ID	CUSTOMER_NAME	CUSTOMER_TYPE
1	CUSTOMER_1	CC
2	CUSTOMER_2	I
3	CUSTOMER_3	SM
4	CUSTOMER_4	CC

Contracts table

CONTRACT_ID	CUSTOMER_ID	AMOUNT
1	1	400
2	2	500
3	3	700
4	1	1000
5	2	1200
6	4	900
7	3	2000
8	2	1500

```
CREATE VIEW CustomersList
AS
SELECT c.customer_id, c.customer_name, ctr.contract_id, ctr.amount
FROM customers c, contracts ctr
WHERE c.customer_id = ctr.customer_id
AND c.customer_type='CC' ;
```

Result:

Customer_Id	Customer_Name	Contract_Id	Amount
1	CUSTOMER_1	1	400
1	CUSTOMER_1	4	1000
4	CUSTOMER_4	6	900

Alter View

Alter View Example

```
CREATE VIEW CustomersList
AS
SELECT c.customer_id, c.customer_name, ctr.contract_id, ctr.amount
FROM customers c, contracts ctr
WHERE c.customer_id = ctr.customer_id
AND c.customer_type='CC' ;
```

Customer_Id	Customer_Name	Contract_Id	Amount
1	CUSTOMER_1	1	400
1	CUSTOMER_1	4	1000
4	CUSTOMER_4	6	900

```
ALTER VIEW CustomersList
AS
SELECT c.customer_id, c.customer_name, ctr.amount
FROM customers c, contracts ctr
WHERE c.customer_id = ctr.customer_id
AND c.customer_type='CC' ;
```

Customer_Id	Customer_Name	Amount
1	CUSTOMER_1	400
1	CUSTOMER_1	1000
4	CUSTOMER_4	900

Modify Data From View

Modify Data From View Example

```
CREATE VIEW CountriesView
AS
SELECT * FROM countries ;

INSERT INTO CountriesView(Country_Id, Country_Code, Country_Name)
VALUES (8, 'ESP', 'SPAIN');
```

Country_Id	Country_Code	Country_Name
1	US	UNITED STATES
2	ENG	ENGLAND
3	FRA	FRANCE
4	DEU	GERMANY
5	CAN	CANADA
6	AUS	AUSTRALIA
7	JPN	JAPAN
8	ESP	SPAIN

```
UPDATE CountriesView
SET Country_Code='ITA', Country_Name='ITALY'
WHERE Country_Id=8;

DELETE FROM CountriesView WHERE Country_Id=8;
```

Rename View

Rename View Syntax

```
sp_rename 'OLD_VIEW', 'NEW_VIEW';
```

Rename View Example

```
sp_rename 'CountriesView', 'NewCountriesView' ;
```

Drop View

Drop View Syntax

DROP VIEW View_Name;

Drop View Example

CREATE VIEW CountriesView
AS
SELECT * FROM countries ;

DROP VIEW CountriesView;