T-SQL

Transact SQL

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

T-SQL (Transact-SQL) is an extension of SQL language. This tutorial covers the fundamental concepts of T-SQL such as its various functions, procedures, indexes, and transactions related to the topic. Each topic is explained using examples for easy understanding.

Audience

This tutorial is designed for those who want to learn the basics of T-SQL.

Prerequisites

To go ahead with this tutorial, familiarity with database concepts is preferred. It is good to have SQL Server installed on your computer, as it might assist you in executing the examples yourself and get to know how it works.

Disclaimer & Copyright

Table of Contents

1. T-SQL – Overview

In 1970's the product called 'SEQUEL', structured English query language, developed by IBM and later SEQUEL was renamed to 'SQL' which stands for Structured Query Language.

In 1986, SQL was approved by ANSI (American national Standards Institute) and in 1987, it was approved by ISO (International Standards Organization).

SQL is a structure query language which is a common database language for all RDBMS products. Different RDBMS product vendors have developed their own database language by extending SQL for their own RDBMS products.

T-SQL stands for Transact Structure Query Language which is a Microsoft product and is an extension of SQL Language.

Example

MS SQL Server - SQL\T-SQL

ORACLE - SQL\PL-SQL

2. T-SQL Server – Data Types

SQL Server data type is an attribute that specifies types of data of any object. Each column, variable and expression has related data type in SQL Server. These data types can be used while creating tables. You can choose a particular data type for a table column based on your requirement.

SQL Server offers seven categories including other category of data types for use.

Exact Numeric Types

Type	From	To
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 –1
numeric	-10^38 +1	10^38 –1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Numeric and decimal are Fixed precision and scale data types and are functionally equivalent.

Approximate Numeric Types

Type	From	To
Float	-1.79E + 308	1.79E + 308
Real	-3.40E + 38	3.40E + 38

Date and Time Types

Type	From	To
datetime (3.33 milliseconds accuracy)	Jan 1, 1753	Dec 31, 9999

smalldatetime (1 minute accuracy)	Jan 1, 1900	Jun 6, 2079
date (1 day accuracy. Introduced in SQL Server 2008)	Jan 1, 0001	Dec 31, 9999
datetimeoffset (100 nanoseconds accuracy. Introduced in SQL Server 2008)	Jan 1, 0001	Dec 31, 9999
datetime2 (100 nanoseconds accuracy. Introduced in **SQL Server 2008**)	Jan 1, 0001	Dec 31, 9999
time (100 nanoseconds accuracy. Introduced in **SQL Server 2008**)	00:00:00.0000000	23:59:59.9999999

Character Strings

Type	Description
char	Fixed-length non-Unicode character data with a maximum length of 8,000 characters.
varchar	Variable-length non-Unicode data with a maximum of 8,000 characters.
Varchar (max)	Variable-length non-Unicode data with a maximum length of 2^{31} characters (Introduced in SQL Server 2005).
text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings

Type	Description
nchar	Fixed-length Unicode data with a maximum length of 4,000 characters.
nvarchar	Variable-length Unicode data with a maximum length of 4,000 characters.
Nvarchar (max)	Variable-length Unicode data with a maximum length of 2^{30} characters (Introduced in SQL Server 2005).

ntext	Variable-length Unicode data with a maximum length of 1,073,741,823 characters.

Binary Strings

Type	Description
binary	Fixed-length binary data with a maximum length of 8,000 bytes.
varbinary	Variable-length binary data with a maximum length of 8,000 bytes.
varbinary(max)	Variable-length binary data with a maximum length of 2^{31} bytes (Introduced in SQL Server 2005).
image	Variable-length binary data with a maximum length of 2,147,483,647 bytes.

Other Data Types

- **sql_variant**: Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
- **timestamp**: Stores a database-wide unique number that gets updated every time a row gets updated.
- **uniqueidentifier**: Stores a globally unique identifier (GUID).
- **xml**: Stores XML data. You can store XML instances in a column or a variable (Introduced in SQL Server 2005).
- **cursor**: A reference to a cursor.
- **table**: Stores a result set for later processing.
- **hierarchyid**: A variable length, system data type used to represent position in a hierarchy (Introduced in SQL Server 2008).

3. T-SQL Server – Create Tables

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL Server **CREATE TABLE** statement is used to create a new table.

Syntax

Following is the basic syntax of CREATE TABLE statement:

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
   PRIMARY KEY( one or more columns ));
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement. Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer to understand with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check complete details at Create Table Using another Table.

Example

In this example, let's create a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table:

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID));
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the following command:

```
exec sp_columns CUSTOMERS
```

The above command produces the following output.

```
TABLE_QUALIFIER    TABLE_OWNER  TABLE_NAME   COLUMN_NAME  DATA_TYPE
     TYPE_NAME     PRECISION    LENGTH SCALE  RADIX  NULLABLE     REMARKS
     COLUMN_DEF    SQL_DATA_TYPESQL_DATETIME_SUB    CHAR_OCTET_LENGTH
     ORDINAL_POSITION    IS_NULLABLE  SS_DATA_TYPE


TestDB      dbo    CUSTOMERS    ID     4      int    10     4      0      10     0
     NULL   NULL   4      NULL   NULL   1      NO     56


TestDB      dbo    CUSTOMERS    NAME   12     varchar       20     20     NULL
     NULL   0      NULL   NULL   12     NULL   20     2      NO     39


TestDB      dbo    CUSTOMERS    AGE    4      int    10     4      0      10     0
     NULL   NULL   4      NULL   NULL   3      NO     56


TestDB      dbo    CUSTOMERS    ADDRESS       1      char   25     25     NULL
     NULL   1      NULL   NULL   1      NULL   25     4      YES    39


TestDB      dbo    CUSTOMERS    SALARY 3      decimal       18     20     2      10
     1      NULL   NULL   3      NULL   NULL   5      YES    106
```

You can now see that CUSTOMERS table is available in your database which you can use to store required information related to customers.

4. T-SQL Server – Drop Tables

The SQL Server **DROP TABLE** statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

Note: You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

Syntax

Following is the basic syntax of DROP TABLE statement:

```
DROP TABLE table_name;
```

Example

Let us first verify CUSTOMERS table and then we will delete it from the database:

```
Exec sp_columns CUSTOMERS;
```

The above command shows the following table.

```
TABLE_QUALIFIER    TABLE_OWNER  TABLE_NAME   COLUMN_NAME  DATA_TYPE
     TYPE_NAME     PRECISION    LENGTH SCALE  RADIX  NULLABLE     REMARKS
     COLUMN_DEF    SQL_DATA_TYPESQL_DATETIME_SUB    CHAR_OCTET_LENGTH
     ORDINAL_POSITION    IS_NULLABLE  SS_DATA_TYPE


TestDB      dbo    CUSTOMERS    ID     4      int    10     4      0      10     0
     NULL   NULL   4      NULL   NULL   1      NO     56

TestDB      dbo    CUSTOMERS    NAME   12     varchar       20     20     NULL
     NULL   0      NULL   NULL   12     NULL   20     2      NO     39

TestDB      dbo    CUSTOMERS    AGE    4      int    10     4      0      10     0
     NULL   NULL   4      NULL   NULL   3      NO     56


TestDB      dbo    CUSTOMERS    ADDRESS       1      char   25     25     NULL
     NULL   1      NULL   NULL   1      NULL   25     4      YES    39


TestDB      dbo    CUSTOMERS    SALARY 3      decimal       18     20     2      10
     1      NULL   NULL   3      NULL   NULL   5      YES    106
```

CUSTOMERS table is available in the database, so let us drop it. Following is the command for the same.

```
 DROP TABLE CUSTOMERS;
Command(s) completed successfully.
```

With the above command, you will not get any rows.

```
Exec sp_columns CUSTOMERS;
No rows\data will be displayed
```

5. T-SQL Server – INSERT Statement

The SQL Server **INSERT INTO** statement is used to add new rows of data to a table in the database.

Syntax

Following are the two basic syntaxes of INSERT INTO statement.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Where column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You need not specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. Following is the SQL INSERT INTO syntax:

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

Following statements will create six records in CUSTOMERS table:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in CUSTOMERS table using second syntax as follows:

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements will produce the following records in CUSTOMERS table:

```
ID   NAME       AGE      ADDRESS                    SALARY
1    Ramesh     32        Ahmedabad                 2000.00
2    Khilan     25        Delhi                     1500.00
3    kaushik    23        Kota                     2000.00
4    Chaitali   25        Mumbai                    6500.00
5    Hardik    27         Bhopal                    8500.00
6    Komal     22          MP                       4500.00
7    Muffy     24        Indore                    10000.00
```

Populate One Table Using Another Table

You can populate data into a table through SELECT statement over another table provided another table has a set of fields, which are required to populate first table. Following is the syntax:

```
INSERT INTO first_table_name
 SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

6. T-SQL Server – SELECT Statement

SQL Server **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called **result-sets**.

Syntax

Following is the basic syntax of SELECT statement:

```
SELECT column1, column2, columnN FROM table_name;
```

Where, column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax:

```
SELECT * FROM table_name;
```

Example

Consider the CUSTOMERS table having the following records:

```
ID    NAME          AGE         ADDRESS       SALARY
1     Ramesh         32         Ahmedabad     2000.00
2     Khilan         25          Delhi         1500.00
3     kaushik       23           Kota         2000.00
4     Chaitali      25           Mumbai        6500.00
5     Hardik        27          Bhopal         8500.00
6     Komal         22           MP             4500.00
7     Muffy          24         Indore         10000.00
```

Following command is an example, which would fetch ID, Name and Salary fields of the customers available in CUSTOMERS table:

```
SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

The above command will produce the following output.

```
ID    NAME        SALARY
1     Ramesh      2000.00
2     Khilan       1500.00
3     kaushik       2000.00
4     Chaitali      6500.00
5     Hardik       8500.00
```

```
6    Komal  4500.00
7    Muffy  10000.00
```

If you want to fetch all the fields of CUSTOMERS table, then use the following query:

```
SELECT * FROM CUSTOMERS;
```

The above will produce the following output.

```
ID   NAME         AGE    ADDRESS      SALARY
1    Ramesh       32     Ahmedabad    2000.00
2    Khilan       25     Delhi         1500.00
3    kaushik       23    Kota         2000.00
4    Chaitali      25    Mumbai        6500.00
5    Hardik        27    Bhopal         8500.00
6    Komal        22     MP             4500.00
7    Muffy        24     Indore         10000.00
```

7. T-SQL Server – UPDATE Statement

The SQL Server **UPDATE** Query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

Syntax

Following is the basic syntax of UPDATE query with WHERE clause:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command is an example, which would update ADDRESS for a customer whose ID is 6:

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

CUSTOMERS table will now have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Pune	4500.00
7	Muffy	24	Indore	10000.00

If you want to modify all ADDRESS and SALARY column values in CUSTOMERS table, you do not need to use WHERE clause. UPDATE query would be as follows:

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

CUSTOMERS table will now have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1000.00
2	Khilan	25	Pune	1000.00
3	kaushik	23	Pune	1000.00
4	Chaitali	25	Pune	1000.00
5	Hardik	27	Pune	1000.00
6	Komal	22	Pune	1000.00
7	Muffy	24	Pune	1000.00

8. T-SQL Server – DELETE Statement

The SQL Server **DELETE** Query is used to delete the existing records from a table.

You have to use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax

Following is the basic syntax of DELETE query with WHERE clause:

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records:

```
ID   NAME          AGE   ADDRESS          SALARY
1    Ramesh        32    Ahmedabad        2000.00
2    Khilan        25    Delhi             1500.00
3    kaushik       23    Kota              2000.00
4    Chaitali      25    Mumbai            6500.00
5    Hardik        27    Bhopal             8500.00
6    Komal         22    MP                 4500.00
7    Muffy         24    Indore             10000.00
```

Following command is an example, which would DELETE a customer, whose ID is 6:

```
DELETE FROM CUSTOMERS
WHERE ID = 6;
```

CUSTOMERS table will now have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause. DELETE query would be as follows:

```
DELETE FROM CUSTOMERS;
```

CUSTOMERS table now will not have any record.

9. T-SQL Server – WHERE Clause

The MS SQL Server **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied, only then it returns a specific value from the table. You will have to use WHERE clause to filter the records and fetch only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

Syntax

Following is the basic syntax of SELECT statement with WHERE clause:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using comparison or logical operators like >, <, =, LIKE, NOT, etc. The following example will make this concept clear.

Example

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

The above command will produce the following output.

```
ID   NAME       SALARY
4    Chaitali    6500.00
5    Hardik     8500.00
6    Komal      4500.00
7    Muffy     10000.00
```

Following command is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with the name 'Hardik'. It is important to note that all the strings should be given inside single quotes ('') whereas numeric values should be given without any quote as in the above example:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

The above command will produce the following output.

```
ID   NAME     SALARY
5    Hardik   8500.00
```

10. T-SQL Server – LIKE Clause

The MS SQL Server **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax

Following is the basic syntax of % and _.

```
SELECT *\column-list FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT *\column-list FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT *\column-list FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT *\column-list FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT  *\column-list FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. XXXX could be any numeric or string value.

Example

Following are a number of examples showing WHERE part having different LIKE clause with '%' and '_' operators.

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command is an example, which will display all the records from CUSTOMERS table where SALARY starts with 200.

```
SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

The above command will produce the following output.

```
ID    NAME          AGE    ADDRESS              SALARY
1     Ramesh 32     Ahmedabad                 2000.00
3     kaushik       23     Kota                 2000.00
```

11. T-SQL Server – ORDER BY Clause

The MS SQL Server ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sort query results in ascending order by default.

Syntax

Following is the basic syntax of ORDER BY clause.

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example

Consider the CUSTOMERS table having the following records:

```
ID   NAME          AGE   ADDRESS            SALARY
1    Ramesh         32   Ahmedabad           2000.00
2    Khilan         25   Delhi               1500.00
3    kaushik        23   Kota                2000.00
4    Chaitali       25   Mumbai              6500.00
5    Hardik         27   Bhopal              8500.00
6    Komal          22   MP                  4500.00
7    Muffy          24   Indore             10000.00
```

Following command is an example, which would sort the result in ascending order by NAME and SALARY.

```
SELECT * FROM CUSTOMERS
     ORDER BY NAME, SALARY
```

The above command will produce the following output.

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

Following command is an example, which would sort the result in descending order by NAME.

```
SELECT * FROM CUSTOMERS
     ORDER BY NAME DESC
```

The above command will produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
7	Muffy	24	Indore	10000.00
6	Komal	22	MP	4500.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
4	Chaitali	25	Mumbai	6500.00

12. T-SQL Server – GROUP BY Clause

The SQL Server GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

Following is the basic syntax of GROUP BY clause. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example

Consider the CUSTOMERS table is having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

If you want to know the total amount of salary on each customer, then following will be the GROUP BY query.

```
SELECT NAME, SUM(SALARY) as [sum of salary] FROM CUSTOMERS
     GROUP BY NAME;
```

The above command will produce the following output.

```
NAME      sum of salary
Chaitali    6500.00
Hardik      8500.00
kaushik     2000.00
Khilan      1500.00
Komal       4500.00
Muffy      10000.00
Ramesh      2000.00
```

Let us now consider the following CUSTOMERS table having the following records with duplicate names.

```
ID    NAME          AGE    ADDRESS                SALARY
1     Ramesh         32    Ahmedabad               2000.00
2     Ramesh        25     Delhi                   1500.00
3     kaushik       23     Kota                    2000.00
4     Kaushik       25     Mumbai                  6500.00
5     Hardik        27     Bhopal                  8500.00
6     Komal         22     MP                      4500.00
7     Muffy         24     Indore                  10000.00
```

If we want to know the total amount of salary on each customer, then following will be GROUP BY query.

```
SELECT NAME, SUM(SALARY) as [sum of salary] FROM CUSTOMERS
     GROUP BY NAME
```

The above command will produce the following output.

```
NAME              sum of salary
Hardik             8500.00
kaushik           8500.00
Komal             4500.00
Muffy             10000.00
Ramesh            3500.00
```

13. T-SQL Server – DISTINCT Clause

The MS SQL Server DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax

Following is the basic syntax of DISTINCT keyword to eliminate duplicate records.

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

Example

Consider the CUSTOMERS table having the following records.

```
ID    NAME      AGE    ADDRESS            SALARY
1     Ramesh    32     Ahmedabad           2000.00
2     Khilan     25    Delhi               1500.00
3     kaushik   23     Kota                2000.00
4     Chaitali  25    Mumbai              6500.00
5     Hardik    27     Bhopal              8500.00
6     Komal     22     MP                  4500.00
7     Muffy     24    Indore              10000.00
```

Let us see how the following SELECT query returns duplicate salary records.

```
SELECT SALARY FROM CUSTOMERS
     ORDER BY SALARY
```

The above command will produce the following output where salary 2000 comes twice which is a duplicate record from the original table.

```
SALARY
1500.00
2000.00
2000.00
4500.00
6500.00
8500.00
10000.00
```

Let us now use DISTINCT keyword with the above SELECT query and see the result.

```
SELECT DISTINCT SALARY FROM CUSTOMERS
    ORDER BY SALARY
```

The above command produces the following output where we do not have any duplicate entry.

```
SALARY
1500.00
2000.00
4500.00
6500.00
8500.00
10000.00
```

14. T-SQL Server – Joining Tables

The MS SQL Server Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) CUSTOMERS table is as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

(b) Another table is ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
100	2009-10-08 00:00:00.000	3	1500.00
101	2009-11-20 00:00:00.000	2	1560.00
102	2009-10-08 00:00:00.000	3	3000.00
103	2008-05-20 00:00:00.000	4	2060.00

Let us join these two tables in our SELECT statement as follows:

```
SELECT ID, NAME, AGE, AMOUNT
        FROM CUSTOMERS, ORDERS
        WHERE  CUSTOMERS.ID = ORDERS.CUSTOMER_ID
OR
SELECT A.ID, A.NAME, A.AGE, B.AMOUNT
        FROM CUSTOMERS A inner join  ORDERS B on A.ID=B.Customer_ID
```

The above command will produce the following output.

ID	NAME	AGE	AMOUNT
2	Khilan	25	1560.00
3	kaushik	23	1500.00
3	kaushik	23	3000.00
4	Chaitali	25	2060.00

It is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

MS SQL Server Join Types:

There are different types of joins available in MS SQL Server:

- **INNER JOIN**: Returns rows when there is a match in both tables.
- **LEFT JOIN**: Returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN**: Returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN**: Returns rows when there is a match in one of the tables.
- **SELF JOIN**: This is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the MS SQL Server statement.
- **CARTESIAN JOIN**: Returns the Cartesian product of the sets of records from the two or more joined tables.

15. T-SQL Server – Sub-Queries

A **sub-query** or **Inner query** or **Nested query** is a query within another SQL Server query and embedded within the WHERE clause. A sub query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Sub queries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that sub queries must follow:

- You must enclose a subquery in parenthesis.
- A subquery must include a SELECT clause and a FROM clause.
- A subquery can include optional WHERE, GROUP BY, and HAVING clauses.
- A subquery cannot include COMPUTE or FOR BROWSE clauses.
- You can include an ORDER BY clause only when a TOP clause is included.
- You can nest sub queries up to 32 levels.

Subqueries with SELECT Statement

Subqueries are most frequently used with the SELECT statement. Following is the basic syntax.

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
       (SELECT column_name [, column_name ]
       FROM table1 [, table2 ]
       [WHERE])
```

Example

Consider the CUSTOMERS table having the following records.

```
ID    NAME          AGE    ADDRESS              SALARY
1     Ramesh 32     Ahmedabad                 2000.00
2     Khilan        25     Delhi                     1500.00
3     kaushik       23     Kota                      2000.00
4     Chaitali      25     Mumbai                    6500.00
5     Hardik        27     Bhopal                    8500.00
6     Komal         22     MP                        4500.00
```

```
7    Muffy        24    Indore                    10000.00
```

Let us apply the following subquery with SELECT statement.

```
SELECT *
     FROM CUSTOMERS
     WHERE ID IN (SELECT ID
                  FROM CUSTOMERS
                  WHERE SALARY > 4500)
```

The above command will produce the following output.

```
ID   NAME         AGE   ADDRESS            SALARY
4    Chaitali     25    Mumbai             6500.00
5    Hardik       27    Bhopal             8500.00
7    Muffy        24    Indore              10000.00
```

Subqueries with INSERT Statement

Sub queries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date, or number functions.

Following is the basic syntax.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
           SELECT [ *|column1 [, column2 ]
           FROM table1 [, table2 ]
           [ WHERE VALUE OPERATOR ]
```

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Following is the syntax to copy complete CUSTOMERS table into CUSTOMERS_BKP.

```
 INSERT INTO CUSTOMERS_BKP
     SELECT * FROM CUSTOMERS
     WHERE ID IN (SELECT ID
                  FROM CUSTOMERS)
```

Subqueries with UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

Following is the basic syntax.

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example

Let us assume we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.

Following command example updates SALARY by 0.25 times in CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
 UPDATE CUSTOMERS
     SET SALARY = SALARY * 0.25
     WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
                   WHERE AGE >= 27 )
```

This will impact two rows and finally CUSTOMERS table will have the following records.

```
ID    NAME          AGE    ADDRESS                SALARY
1     Ramesh         32    Ahmedabad               500.00
2     Khilan         25    Delhi                  1500.00
3     kaushik        23    Kota                   2000.00
4     Chaitali       25    Mumbai                 6500.00
5     Hardik         27    Bhopal                 2125.00
6     Komal          22    MP                     4500.00
7     Muffy          24    Indore                10000.00
```

Subqueries with DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

Following is the basic syntax.

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example

Let us assume we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.

Following command example deletes records from CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
DELETE FROM CUSTOMERS
     WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
                   WHERE AGE >=27 )
```

This would impact two rows and finally CUSTOMERS table will have the following records.

```
ID    NAME          AGE    ADDRESS                  SALARY
2     Khilan        25     Delhi                    1500.00
3     kaushik       23     Kota                     2000.00
4     Chaitali      25     Mumbai                   6500.00
6     Komal         22     MP                       4500.00
7     Muffy         24     Indore                   10000.00
```

16. T-SQL Server – Stored Procedures

The MS SQL Server **Stored procedure** is used to save time to write code again and again by storing the same in database and also get the required output by passing parameters.

Syntax

Following is the basic syntax of Stored procedure creation.

```
Create procedure <procedure_Name>
As
Begin
<SQL Statement>
End
Go
```

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command is an example which would fetch all records from the CUSTOMERS table in Testdb database.

```
CREATE PROCEDURE SelectCustomerstabledata
AS
SELECT * FROM Testdb.Customers
GO
```

The above command will produce the following output.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

17. T-SQL Server – Transactions

A **transaction** is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym ACID:

- **Atomicity**: Ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
- **Consistency**: Ensures that the database properly changes state upon a successfully committed transaction.
- **Isolation**: Enables transactions to operate independently of and transparent to each other.
- **Durability**: Ensures that the result or effect of a committed transaction persists in case of a system failure.

Transaction Control

There are following commands used to control transactions:

- **COMMIT**: To save the changes.
- **ROLLBACK**: To roll back the changes.
- **SAVEPOINT**: Creates points within groups of transactions in which to ROLLBACK.
- **SET TRANSACTION**: Places a name on a transaction.

Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

In order to use transactional control commands in MS SQL Server, we have to begin transaction with 'begin tran' or begin transaction command otherwise these commands will not work.

COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. This command saves all transactions to the database since the last COMMIT or ROLLBACK command.

Following is the syntax for COMMIT command.

```
COMMIT;
```

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command example will delete records from the table having age = 25 and then COMMIT the changes in the database.

```
Begin Tran
DELETE FROM CUSTOMERS
     WHERE AGE = 25
COMMIT
```

As a result, two rows from the table would be deleted and SELECT statement will produce the following output.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Following is the syntax for ROLLBACK command.

```
ROLLBACK
```

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command example will delete records from the table having age = 25 and then ROLLBACK the changes in the database.

```
Begin Tran
DELETE FROM CUSTOMERS
     WHERE AGE = 25;
ROLLBACK
```

As a result, delete operation will not impact the table and SELECT statement will produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SAVEPOINT Command

SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

Following is the syntax for SAVEPOINT command.

```
SAVE TRANSACTION SAVEPOINT_NAME
```

This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions.

Following is the syntax for rolling back to a SAVEPOINT.

```
ROLLBACK TO SAVEPOINT_NAME
```

In the following example, we will delete three different records from the CUSTOMERS table. We will have to create a SAVEPOINT before each delete, so that we can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following are the series of operations:

```
Begin Tran
SAVE Transaction SP1
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=1
1 row deleted.
SAVE Transaction SP2
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=2
1 row deleted.
```

```
SAVE Transaction SP3
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=3
1 row deleted.
```

The three deletions have taken place, however, we have changed our mind and decide to ROLLBACK to the SAVEPOINT that we identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:

```
ROLLBACK Transaction SP2
Rollback complete.
```

Notice that only the first deletion took place since we rolled back to SP2.

```
SELECT * FROM CUSTOMERS
```

6 rows selected.

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Ahmedabad	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SET TRANSACTION Command

SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows.

Following is the syntax for SET TRANSACTION.

```
SET TRANSACTION ISOLATION LEVEL <Isolationlevel_name>
```

18. T-SQL Server – Indexes

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an **index** is a pointer to data in a table. An index in a database is very similar to an index at the end of a book.

For example, if you want to reference all the pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

CREATE INDEX Command

Following is the basic syntax of CREATE INDEX.

```
CREATE INDEX index_name ON table_name
```

Single-Column Indexes

A single-column index is one that is created based on only one table column. Following is the basic syntax.

```
CREATE INDEX index_name
ON table_name (column_name)
```

Example

```
CREATE INDEX singlecolumnindex
ON customers (ID)
```

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. Following is the basic syntax.

```
CREATE UNIQUE INDEX index_name
on table_name (column_name)
```

Example

```
CREATE UNIQUE INDEX uniqueindex
on customers (NAME)
```

Composite Indexes

A composite index is an index on two or more columns of a table. Following is the basic syntax.

```
CREATE INDEX index_name
on table_name (column1, column2)
```

Example

```
CREATE INDEX compositeindex
on customers (NAME, ID)
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

DROP INDEX Command

An index can be dropped using MS SQL SERVER DROP command. Care should be taken when dropping an index because performance may be slowed or improved.

Following is the basic syntax.

```
DROP INDEX tablename.index_name
```

When to Avoid Indexes?

Although indexes are intended to enhance the performance of databases, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

- Indexes should not be used on small tables.
- Tables that have frequent, large batch update or insert operations should not be indexed.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

19. T-SQL Server – SQL Functions

MS SQL Server has many built-in functions to perform processing on string or numeric data. Following is the list of all useful SQL built-in functions:

- **SQL Server COUNT Function** - The SQL Server COUNT aggregate function is used to count the number of rows in a database table.
- **SQL Server MAX Function** - The SQL Server MAX aggregate function allows to select the highest (maximum) value for a certain column.
- **SQL Server MIN Function** - The SQL Server MIN aggregate function allows to select the lowest (minimum) value for a certain column.
- **SQL Server AVG Function** - The SQL Server AVG aggregate function selects the average value for certain table column.
- **SQL Server SUM Function** - The SQL Server SUM aggregate function allows selecting the total for a numeric column.
- **SQL Server SQRT Function** - This is used to generate a square root of a given number.
- **SQL Server RAND Function** - This is used to generate a random number using SQL command.
- **SQL Server CONCAT Function** - This is used to concatenate multiple parameters to a single parameter.
- **SQL Server Numeric Functions** - Complete list of SQL functions required to manipulate numbers in SQL.
- **SQL Server String Functions** - Complete list of SQL functions required to manipulate strings in SQL.

20. T-SQL Server – String Functions

MS SQL Server String functions can be applied on string value or will return string value or numeric data.

Following is the list of String functions with examples.

ASCII()

Ascii code value will come as output for a character expression.

Example: The following query will give the Ascii code value of a given character.

```
Select ASCII ('word')
```

CHAR()

Character will come as output for given Ascii code or integer.

Example: The following query will give the character for a given integer.

```
Select CHAR(97)
```

NCHAR()

Unicode character will come as output for a given integer.

Example: The following query will give the Unicode character for a given integer.

```
Select NCHAR(300)
```

CHARINDEX()

Starting position for given search expression will come as output in a given string expression.

Example: The following query will give the starting position of 'G' character for given string expression 'KING'.

```
Select CHARINDEX('G', 'KING')
```

LEFT()

Left part of the given string till the specified number of characters will come as output for a given string.

Example: The following query will give the 'WORL' string as mentioned 4 number of characters for given string 'WORLD'.

```
Select LEFT('WORLD', 4)
```

RIGHT()

Right part of the given string till the specified number of characters will come as output for a given string.

Example: The following query will give the 'DIA' string as mentioned 3 number of characters for given string 'INDIA'.

```
Select RIGHT('INDIA', 3)
```

SUBSTRING()

Part of a string based on the start position value and length value will come as output for a given string.

Example: The following queries will give the 'WOR', 'DIA', 'ING' strings as we mentioned (1,3), (3,3) and (2,3) as start and length values respectively for given strings 'WORLD', 'INDIA' and 'KING'.

```
Select SUBSTRING ('WORLD', 1,3)
Select SUBSTRING ('INDIA', 3,3)
Select SUBSTRING ('KING', 2,3)
```

LEN()

Number of characters will come as output for a given string expression.

Example: The following query will give the 5 for the 'HELLO' string expression.

```
Select LEN('HELLO')
```

LOWER()

Lowercase string will come as output for a given string data.

Example: The following query will give the 'sqlserver' for the 'SQLServer' character data.

```
Select LOWER('SQLServer')
```

UPPER()

Uppercase string will come as output for a given string data.

Example: The following query will give the 'SQLSERVER' for the 'SqlServer' character data.

```
Select UPPER('SqlServer')
```

LTRIM()

String expression will come as output for a given string data after removing leading blanks.

Example: The following query will give the 'WORLD' for the '   WORLD' character data.

```
Select LTRIM('   WORLD')
```

RTRIM()

String expression will come as output for a given string data after removing trailing blanks.

Example: The following query will give the 'INDIA' for the 'INDIA   ' character data.

```
Select RTRIM('INDIA   ')
```

REPLACE()

String expression will come as output for a given string data after replacing all occurrences of specified character with specified character.

Example: The following query will give the 'KNDKA' string for the 'INDIA' string data.

```
Select REPLACE('INDIA', 'I', 'K')
```

REPLICATE()

Repeat string expression will come as output for a given string data with specified number of times.

Example: The following query will give the 'WORLDWORLD' string for the 'WORLD' string data.

```
Select REPLICATE('WORLD', 2)
```

REVERSE()

Reverse string expression will come as output for a given string data.

Example: The following query will give the 'DLROW' string for the 'WORLD' string data.

```
Select REVERSE('WORLD')
```

SOUNDEX()

Returns four-character (SOUNDEX) code to evaluate the similarity of two given strings.

Example: The following query will give the 'S530' for the 'Smith', 'Smyth' strings.

```
Select SOUNDEX('Smith'), SOUNDEX('Smyth')
```

DIFFERENCE()

Integer value will come as output of given two expressions.

Example: The following query will give the 4 for the 'Smith', 'Smyth' expressions.

```
Select Difference('Smith','Smyth')
```

Note: If the output value is 0 it indicates weak or no similarity between give 2 expressions.

SPACE()

String will come as output with the specified number of spaces.

Example: The following query will give the 'I LOVE INDIA'.

```
Select 'I'+space(1)+'LOVE'+space(1)+'INDIA'
```

STUFF()

String expression will come as output for a given string data after replacing from starting character till the specified length with specified character.

Example: The following query will give the 'AIJKFGH' string for the 'ABCDEFGH' string data as per given starting character and length as 2 and 4 respectively and 'IJK' as specified target string.

```
Select STUFF('ABCDEFGH', 2,4,'IJK')
```

STR()

Character data will come as output for the given numeric data.

Example: The following query will give the 187.37 for the given 187.369 based on specified length as 6 and decimal as 2.

```
Select STR(187.369,6,2)
```

UNICODE()

Integer value will come as output for the first character of given expression.

Example: The following query will give the 82 for the 'RAMA' expression.

```
Select UNICODE('RAMA')
```

QUOTENAME()

Given string will come as output with the specified delimiter.

Example: The following query will give the "RAMA" for the given 'RAMA' string as we specified double quote as delimiter.

```
Select QUOTENAME('RAMA','"')
```

PATINDEX()

Starting position of the first occurrence from the given expression as we specified 'I' position is required.

Example: The following query will give the 1 for the 'INDIA'.

```
Select PATINDEX('I%','INDIA')
```

FORMAT()

Given expression will come as output with the specified format.

Example: The following query will give the ' Monday, November 16, 2015' for the getdate function as per specified format with 'D' refers weekday name.

```
SELECT FORMAT ( getdate(), 'D')
```

CONCAT()

Single string will come as output after concatenating the given parameter values.

Example: The following query will give the 'A,B,C' for the given parameters.

```
Select CONCAT('A',',','B',',','C')
```

21. T-SQL Server – Date Functions

Following is the list of date functions in MS SQL Server.

GETDATE()

It will return the current date along with time.

Syntax for the above function:

```
GETDATE()
```

Example: The following query will return the current date along with time in MS SQL Server.

```
Select getdate() as currentdatetime
```

DATEPART()

It will return the part of date or time.

Syntax for the above function:

```
DATEPART(datepart, datecolumnname)
```

Example 1: The following query will return the part of current date in MS SQL Server.

```
Select datepart(day, getdate()) as currentdate
```

Example 2: The following query will return the part of current month in MS SQL Server.

```
Select datepart(month, getdate()) as currentmonth
```

DATEADD()

It will display the date and time by add or subtract date and time interval.

Syntax for the above function:

```
DATEADD(datepart, number, datecolumnname)
```

Example: The following query will return the after 10 days date and time from the current date and time in MS SQL Server.

```
Select dateadd(day, 10, getdate()) as after10daysdatetimefromcurrentdatetime
```

DATEDIFF()

It will display the date and time between two dates.

Syntax for the above function:

```
DATEDIFF(datepart, startdate, enddate)
```

Example: The following query will return the difference of hours between 2015-11-16 and 2015-11-11dates in MS SQL Server.

```
Select datediff(hour, 2015-11-16, 2015-11-11) as
differencehoursbetween20151116and20151111
```

CONVERT()

It will display the date and time in different formats.

Syntax for the above function:

```
CONVERT(datatype, expression, style)
```

Example: The following queries will return the date and time in different format in MS SQL Server.

```
SELECT CONVERT(VARCHAR(19),GETDATE())
SELECT CONVERT(VARCHAR(10),GETDATE(),10)
SELECT CONVERT(VARCHAR(10),GETDATE(),110)
```

22. T-SQL Server – Numeric Functions

MS SQL Server numeric functions can be applied on numeric data and will return numeric data.

Following is the list of Numeric functions with examples.

ABS()

Absolute value will come as output for numeric expression

Example: The following query will give the absolute value.

```
Select ABS(-22)
```

ACOS()

Arc cosine value will come as output for the specified numeric expression.

Example: The following query will give the arc cosine value of 0.

```
Select ACOS(0)
```

ASIN()

Arc sine value will come as output for the specified numeric expression.

Example: The following query will give the arc sine value of 0.

```
Select ASIN(0)
```

ATAN()

Arc tangent value will come as output for the specified numeric expression.

Example: The following query will give the arc tangent value of 0.

```
Select ATAN(0)
```

ATN2()

Arc tangent value in all four quadrants will come as output for the specified numeric expression.

Example: The following query will give the arc tangent value in all four quadrants of 0.

```
Select ATN2(0, -1)
```

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

BETWEEN()

If the values exist between given two expressions then those will be come as output.

Example: The following query will give the following output.

```
SELECT salary from customers where salary between 2000 and 8500
```

Output

```
salary
2000.00
2000.00
6500.00
8500.00
4500.00
```

MIN()

Minimum value will come as output from the given expression.

Example: The following query will give '1500.00' for the given 'salary' expression from the customers table.

```
Select MIN(salary)from CUSTOMERS
```

MAX()

Maximum value will come as output from the given expression.

Example: The following query will give '10000.00' for the given 'salary' expression from the customers table.

```
Select MAX(salary)from CUSTOMERS
```

SQRT()

Square root of the given numeric expression will come as output.

Example: The following query will give 2 for the given 4 numeric expression.

```
Select SQRT(4)
```

PI()

PI value will come as output.

Example: The following query will give 3.14159265358979 for the PI value.

```
Select PI()
```

CEILING()

Given value will come as output after rounding the decimals which is the next highest value.

Example: The following query will give 124 for the given 123.25 value.

```
Select CEILING(123.25)
```

FLOOR()

Given value will come as output after rounding the decimals which is less than or equal to the expression.

Example: The following query will give 123 for the given 123.25 value.

```
Select FLOOR(123.25)
```

LOG()

Natural logarithm of the given expression will come as output.

Example: The following query will give 0 for the given 1 value.

```
Select LOG(1)
```