**SQL TUTOTRIALS**

* **SQL** is a database computer language designed for the retrieval and management of data in a relational database like MySQL, MS Access, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres etc.
* **SQL** stands for **Structured Query Language**.
* SQL was developed in the 1970s by IBM Computer Scientists.
* **SQL** is not a database management system, but it is a query language which is used to store and retrieve the data from a database or in simple words SQL is a language that communicates with databases.

**MS SQL Server**

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are −

* T-SQL
* ANSI SQL

## What are SQL datatypes?

* Datatypes in SQL are used to define the type of data that can be stored in a column of a table, like, INT, CHAR, MONEY, DATETIME etc.
* They provide guidelines for SQL to understand what type of data is expected inside each column, and they also identify how SQL will interact with the stored data.
* The datatype specification, hence, prevents the user from entering any unexpected or invalid data.

**Types of Datatypes**

There are three main types of datatypes in the SQL server. They are listed below −

* **String**
  + Character String
    - Char(n) – max 8000 characters
    - Varchar(n) – max 8000 characters
    - Text(n) – 2,147,483,647 characters
  + Unicode character String
    - Nchar(n) – max 400 characters
    - Nvarchar(n) – max 4000 characters
    - Ntext – 1073,741,823 characters
  + Binary Strings
    - Binary (n) – max 8000 character
    - Varbinary(n) – max 8000 characters47,483,647 charcteres
    - Image – 2,1
* **Numeric**
  + bigInt
  + int
  + smallInt
  + TinyInt
  + Bit
  + Decimal
  + SamllMoney
  + Numeric(x,y)
* **Date and Time**
  + Date
  + Time
  + Smalldatetime
  + DateTime

# **DATABASE**

1.Creating the Database CREATE DATABASE DatabaseName;

2.Switch the database USE DatabaseName;

3.Delete the database DROP DATABASE DatabaseName;

4.Delete the database by conditon DROP DATABASE IF EXISTS DatabaseName;

5.Renaming the database ALTER DATABASE existing\_database\_name MODIFY NAME

=new\_database\_name;

6.Rename database with sp EXEC sp\_renamedb 'existing\_database\_name',

'new\_database\_name';

7.Show all databases SELECT name From sys.databases;

Note: Cannot change the name of the system database master.

## Listing databases in SQL server

SELECT \*

FROM sys.databases;

## Stored procedure to show all the Databases

EXEC sp\_databases;

## Backup database statement in SQL

BACKUP DATABASE database\_name

TO DISK = 'filepath';

Ex. BACKUP DATABASE testDB TO DISK = 'D:\DB\_backup.bak';

## Backup Database with SQL DIFFERENTIAL Statement

BACKUP DATABASE database\_name

TO DISK = 'filepath'

WITH DIFFERENTIAL;

NOTE : The differential backup contains only the changes made to the database since the last full backup.

# **TABLE**

## Create table

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

# **Dropping TABLE Command**

Drop table table\_name;

DROP TABLE [IF EXISTS] table\_name;

## Metadata of table

EXEC sp\_help TableName;

exec sp\_columns CUSTOMERS;

## Using SQL sys.tables view

SELECT \* FROM sys.tables;

## Using SQL schema.tables view

SELECT table\_name, table\_type

FROM information\_schema.tables;

NOTE : Following is the query to retrieve the list of all the tables in the information\_schema present in the database −

## Using SQL sysobjects view

SELECT \* FROM sysobjects;

NOTE: We can use **sysobjects** view to retrieve the information of all the objects created in the database, including stored procedures, views, system tables and user-defined tables.

## Renaming a table using sp\_rename

EXEC sp\_rename 'old\_table\_name', 'new\_table\_name';

## Insert data into table

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

## Deleting specific rows based

DELETE FROM table\_name;

DELETE FROM table\_name

WHERE condition;

DELETE FROM table\_name

WHERE condition1 AND condition2 AND ... conditionN;

**TRUNCATE TABLE Command**

TRUNCATE TABLE table\_name;

NOTE : The SQL **TRUNCATE TABLE** command is used to delete all the records from an existing table by reinitializing the table's structure.

# **CLONE TABLES**

* **Cloning operation** in SQL allows the user to create the exact copy of an existing table along with its definition, which is completely independent from the original table. Thus, if any changes are made to the cloned table, they will not be reflected in the original table.
* This operation comes in handy during testing processes, where there is a need to perform sample testing using the existing database tables.

There are three types of cloning possible using SQL in various RDBMS; they are listed below −

* **Simple Cloning** − Creates a new table without copying any constraints or indexes etc.
  + CREATE TABLE new\_table SELECT \* FROM original\_table;
* **Shallow Cloning** − Creates a new empty table with the same table structure as an existing table.
  + CREATE TABLE new\_table LIKE original\_table;
* **Deep Cloning** − Creates a new table and copies the table structure and data of an existing table to the new table.
  + CREATE TABLE new\_table LIKE original\_table;
  + INSERT INTO new\_table SELECT \* FROM original\_table;

### Steps to Clone a Table

Following are the two general ways to copy a table in an SQL server. To fully clone a table, perform the two methods together.

* Generate the CREATE TABLE script to copy the structure of a table.
* Using the SELECT... INTO statement to copy the data of a table.

# **TEMPORARY TABLES**

* Temporary tables are pretty much what their name describes: they are tables that are created in a database to store data temporarily.
* They can perform operations that are like operations of permanent database tables like Create, Update, Delete, Insert and other operations like Join.
* But these tables will be automatically deleted once the current client session is terminated.
* In addition to that, they can also be explicitly deleted if the users decide to drop them manually.

## Types of Temporary Tables

There are two types of temporary tables which are categorized based on their scope of working −

* Local Temporary Tables
* Global Temporary Tables

### Local Temporary Tables

* A Local Temporary Table is accessible only in the session that has created it.
* It is automatically deleted when the connection that has created it gets closed.
* To create Local Temporary Table, a single “#” is used as the prefix of a table name.
* To manually drop this temporary table by using the “DROP TABLE #temp-table” query.
* There will be Random Numbers are appended to the Name of Table Name.
* If the Temporary Table is created inside the stored procedure, it gets dropped automatically upon the completion of stored procedure execution.

### Global Temporary Tables

* Global Temporary Tables are visible to all connections and Dropped when the last connection referencing the table is closed.
* Global Table Name must have a Unique Table Name.
* There will be no random Numbers suffixed at the end of the Table Name.

### Creation Temporary Tables

CREATE TEMPORARY TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

### Dropping Temporary Tables

DROP TABLE SALESSUMMARY;

NOTE : By default, all the temporary tables are deleted by MySQL when your database connection gets terminated. Still if you want to delete them in between, then you can do so by issuing a **DROP TABLE** command.

# **ALTER TABLE COMMAND**

## Adding a new column

ALTER TABLE table\_name ADD column\_name datatype;

## Adding a column after an existing column

ALTER TABLE table\_name

ADD COLUMN (column\_name column\_definition...)

AFTER existing\_column;

## Adding a column at start

ALTER TABLE table\_name ADD COLUMN (column\_name column\_definition...) FIRST;

## Dropping a column

ALTER TABLE table\_name

DROP COLUMN column\_name;

## Adding a primary key

ALTER TABLE table\_name

ADD CONSTRAINT pk\_name PRIMARY KEY (column\_name);

## Dropping a Primary Key

ALTER TABLE table\_name DROP PRIMARY key\_name;

## Adding a foreign key

ALTER TABLE table\_name

ADD CONSTRAINT key FORIEGN KEY (column\_name);

NOTE : For stablish connection between the two tables

ALTER TABLE child\_table\_name

ADD CONSTRAINT key FORIEGN KEY (column\_name)

REFERENCES Parent\_Table\_name(primary\_key\_column);

## Dropping a foreign key

ALTER TABLE table\_name DROP FOREIGN KEY key\_name;

## Adding a constraint

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name UNIQUE(column1, column2...);

## Dropping a constraint

ALTER TABLE EMPLOYEES

DROP CONSTRAINT Constraint\_name;

# **SELECT QUERY**

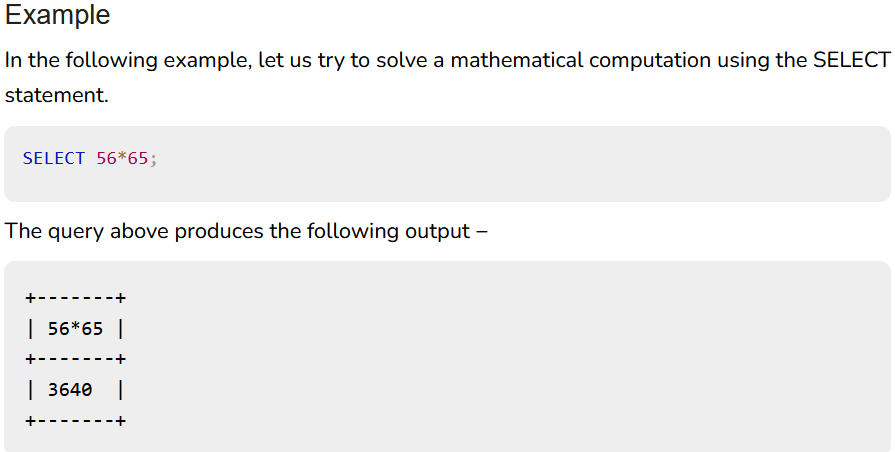
SELECT \* FROM table\_name;

SELECT table\_name.\*, expression1,expression2,…..,expressionN

FROM table\_name;

SELECT column1, column2, columnN FROM table\_name;

SELECT math\_computation;



SELECT column\_name AS alias\_name FROM table\_name;

# **SELECT INTO QUERY**

SELECT \*

INTO new\_table\_name

FROM existing\_table\_name;

SELECT column1, column2, ..., columnN

INTO new\_table\_name

FROM existing\_table\_name;

SELECT column1, column2, ..., columnN

INTO new\_table\_name

FROM table1

JOIN table2 ON table1.column = table2.column;

SELECT \*

INTO new\_table\_name

FROM existing\_table\_name

WHERE condition;

# **INSERT INTO... SELECT**

INSERT INTO table2

SELECT \* FROM table1

**UPDATE QUERY**

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

# **DELETE QUERY**

DELETE FROM table\_name

WHERE [condition];

# **SORTING QUERY**

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1, column2,-- columnN] [ASC | DESC];

**TOP CLAUSE**

SELECT TOP value column\_name(s)

FROM table\_name

WHERE [condition]

SELECT TOP n \*

FROM TableName

ORDER BY AGE ASC/DESC;

SELECT TOP n PERCENT \*

FROM TableName

ORDER BY column\_Name;

SELECT TOP 2 WITH TIES \*

FROM Table\_Name ORDER BY ColumnName;

# **DISTINCT KEYWORD**

SELECT DISTINCT column1, column2,.....columnN

FROM table\_name;

SELECT COUNT(DISTINCT column\_name)

FROM table\_name WHERE condition;

# **ORDER BY CLAUSE**

SELECT column-list

FROM table\_name

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

# **GROUP BY CLAUSE**

SELECT column\_name(s)

FROM table\_name

GROUP BY column\_name(s);

SELECT column1, column2, ..., aggregate\_function(columnX) AS alias

FROM table

GROUP BY column1, column2, ...

ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...;

Graphical user interface

Description automatically generated with medium confidence

SELECT column1, column2, aggregate\_function(column)

FROM table\_name

GROUP BY column1, column2

HAVING condition;

Graphical user interface, text, application

Description automatically generated

## HAVING CLAUSE

SELECT column1, column2, aggregate\_function(column)

FROM table\_name

GROUP BY column1, column2

HAVING condition;

Graphical user interface, text, application

Description automatically generated

# **BOOLEAN (BIT) OPERATOR**

SQL **does not** have a Boolean data type (as a keyword). Instead, it provides BIT data type. A bit data type is an integer value that accepts the values 0, 1 and NULL.

* The value 0 represents FALSE and 1 represents TRUE.
* We can also store NULL values using the bit datatype.
* The range of the bit data type is 1 to 64. This means that **SQL BOOLEAN** requires only a single bit to store values.

NOTE : The databases like PostgreSQL and PL/SQL provides the Boolean data type which is abbreviated as BOOL. Whereas the databases like MySQL and oracle SQL does not have a Boolean data type. To represent Boolean values, they provide TINYINT and BIT data type respectively.

CREATE TABLE table\_name (

column name BIT,

column2 datatype,

column 3 datatype …

);

Example:

Shape, rectangle

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

## Replacing BIT 0,1 with TRUE and FALSE

CASE

WHEN condition1 THEN value1

WHEN condition2 THEN value2

...

ELSE default\_value

END

**Graphical user interface, text, application

Description automatically generated**

**Table

Description automatically generated**

## BIT with stored procedures

CREATE PROCEDURE my\_Procedure

@myBit BIT

AS

BEGIN

IF @myBit = 1

BEGIN

-- do something if @myBit is true

END

ELSE

BEGIN

-- do something if @myBit is false

END

END

Graphical user interface, text, application, email

Description automatically generated

**Table

Description automatically generated**

# **JOINS**

A **Join** is an operation performed on database tables to fetch data from related tables, based on common fields/columns.

There are two types of major joins. They are listed below −

* Cross join or Cartesian join
* Inner Join
  + Theta join
  + Equi join
  + Natural join
* Outer Join
  + Left outer join
  + Right outer join
  + Full outer join
* Self join

**CROSS JOIN OR CARTESIAN JOIN**

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables.

SELECT table1.column1, table2.column2

FROM table1, table2;

SELECT COLUMN1, COLUMN2, COLUMN3,…

FROM TABLE1, TABLE2;

SELECT column\_name(s)

FROM table1 **CROSS JOIN** table2;

**INNER JOIN**

* An Inner Join retrieves the intersection of two tables.
* It compares each row of the first table with each row of the second table.
* If the pairs of these rows satisfy the join-predicate, they are joined together.

SELECT column\_name(s)

FROM table\_name1 **INNER JOIN** table\_name2

ON table\_name1.column\_name = table\_name2.column\_name;

OR

SELECT column\_name(s)

FROM table\_name1 **JOIN** table\_name2

ON table\_name1.column\_name = table\_name2.column\_name;

Typically, the join predicate involves a foreign key from one table and its associated key in the other table.

* The Inner Join is a default join; i.e., even if the “**Join**” keyword is used instead of “**Inner Join**”, tables are joined using matching records of common columns, by default.
* The most important and frequently used join in SQL is the Inner Join. It is also referred to as **Equijoin**.

**OUTER JOIN**

Unlike inner join the outer join may contain the records that doesn’t satisfy the join condition along with the records that satisfy it. There are three types of outer join namely −

Left Outer Join

SELECT column\_name(s)

FROM table\_name1 **LEFT JOIN** table\_name2

ON table\_name1.column\_name = table\_name2.column\_name;

Right Outer Join

SELECT column\_name(s)

FROM table\_name1 **RIGHT JOIN** table\_name2

ON table\_name1.column\_name = table\_name2.column\_name;

Full Outer Join

SELECT column\_name(s)

FROM table\_name1 **FULL OUTER JOIN** table\_name2

ON table\_name1.column\_name = table\_name2.column\_name;

**SELF JOIN**

* **Self Join**, just like its name suggests, is a type of join that combines the records of a table with itself.
* The **SQL Self Join** is used to join a table to itself as if the table were two tables. To carry this out, at least one table is temporarily renamed in the SQL statement.
* Self Join is a type of inner join, which is performed in cases where the comparison between two columns of a same table is required; probably to establish a relationship between them.
* In other words, a table is joined with itself when it contains both **Foreign Key** and **Primary Key** in it.

SELECT column\_name(s)

FROM table1 a, table1 b

WHERE a.common\_field = b.common\_field;

## DELET JOIN

Joins are used to retrieve records from two or more tables, by combining columns of these tables based on the common fields. This merged data can be deleted with all the changes reflected in original tables.

For example, consider a database of an educational institution. It consists of various tables: Departments, Student details, Library passes, Laboratory passes etc. When a set of students are graduated, all their details from the organizational tables need to be removed, as they are unwanted. However, removing the details separately from multiple tables can be cumbersome.

To make this process easier, we will first retrieve the combined data of all graduated students from all the tables using **Joins**; then, this joined data is deleted from all the tables using **DELETE** statement. This entire process can be done in one single query.

DELETE table(s)

FROM table1 JOIN table2

ON table1.common\_field = table2.common\_field;

# **UPDATE JOIN**

Usually, **JOINS** in SQL are used to fetch the combination of rows from multiple tables, with respect to a matching field. And since the **UPDATE** statement only modifies the data in a single table, we combine multiple tables into one using JOINS and then update them. This is also known as **CROSS-TABLE MODIFICATION**.

We know that when a SELECT statement is used with JOINS, it *displays* the contents of multiple tables; similarly, when an UPDATE statement is used with JOINS, it *updates* the contents of multiple tables.

UPDATE table(s)

SET column1 = value1, column2 = value2, ...

FROM table1

JOIN table2 ON column3 = column4;

# **T-SQL – VARIABLE**

A Transact-SQL local variable is a database object that can store a single data value of a specific type.  
**Local variables** : They are declared by the user and start with the '@' symbol. Local variables can be used within a procedure or batch.  
**Global variables** : They are declared by the system beforehand and start with the '@@' symbol. Global variables can store session information.

## Declare a Transact-SQL Variable

**Syntax :**

DECLARE @MyVariable datatype;

**Examples :**   
DECLARE @EMP\_ID INT;  
DECLARE @EMP\_ID AS INT;  
DECLARE @EMP\_NAME VARCHAR (50);  
DECLARE @EMP\_ID AS INT, @EMP\_NAME VARCHAR (50);

## Set a Variable Value

**Syntax :**

SET @Local\_Variable = Value;

**Example :**  
DECLARE @EMP\_ID AS INT;  
SET @EMP\_ID = 5;  
PRINT @EMP\_ID;

## SELECT a Variable

The SELECT statement can be used to select the assigned values by certain criteria as per the requirement of the user. Syntax for the SELECT statement of one variable or multiple variables:

DECLARE @Local\_Variable Data\_Type;  
SET @Local\_Variable = Value;

Example:

DECLARE @EMP\_ID as INT, @EMP\_NAME AS VARCHAR(50);  
SELECT @EMP\_ID = 5, @EMP\_NAME = 'STEVE';  
PRINT @EMP\_ID;  
PRINT @EMP\_NAME;

# **T-SQL – CONDITIONAL STATEMENT**

begin

declare @name varchar(20);

declare @sal numeric(8,2);

declare @grade char(1);

select @name = EmpFName , @sal = Salary

from EMPLOYEE

where EmpCode = 9369;

if @sal >=1000

begin

set @grade = 'A';

end

else if @sal >=1500

begin

set @grade = 'B';

end

else if @sal >=2000

begin

set @grade = 'C';

end

print @name + ' is in grade ' + @grade;

end

select \* from EMPLOYEE;

# **T-SQL – CONDITIONAL STATEMENT**

begin

declare @name varchar(20);

declare @sal numeric(8,2);

declare @grade char(1);

declare @id int = 1;

while @id<=10

begin

select @name = EmpFName , @sal = Salary

from EMPLOYEE

where EmpCode = 9369;

if @sal >=1000

begin

set @grade = 'A';

end

else if @sal >=1500

begin

set @grade = 'B';

end

else if @sal >=2000

begin

set @grade = 'C';

end

print @name + ' is in grade ' + @grade;

set @id= @id +1;

end

end

select \* from EMPLOYEE;

# **T-SQL – 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.

## Properties of Transactions

* **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.
  + COMMIT;
* **ROLLBACK** − To roll back the changes.
  + ROLLBACK;
* **SAVEPOINT** − Creates points within groups of transactions in which to ROLLBACK.
  + SAVE TRANSACTION SAVEPOINT\_NAME;
* **SET TRANSACTION** − Places a name on a transaction.
  + SET TRANSACTION ISOLATION LEVEL <Isolationlevel\_name>;

NOTE : 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.

# **T-SQL - VIEW**

# **Create View**

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

# **To see the all created View in schema**

SELECT TABLE\_SCHEMA,TABLE\_NAME FROM INFORMATION\_SCHEMA.VIEWS;

# **Update View**

UPDATE view\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

# **DROP or DELETE View**

DROP VIEW view\_name;

DROP VIEW [IF EXISTS] view\_name;

# **Rename View**

EXEC sp\_rename 'old\_view\_name', 'new\_view\_name'

# **T-SQL - INDEXES**

Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval.

The basic syntax of a **CREATE INDEX** is as follows.

CREATE INDEX index\_name ON table\_name;

### Clustered Indexes

CREATE clustered index\_name

ON table\_name (column\_name);

### Non-clustered Indexes

CREATE nonclustered INDEX index\_name

ON table\_name (column\_name);

### Single-Column Indexes

CREATE INDEX index\_name

ON table\_name (column\_name);

### Unique Indexes

CREATE UNIQUE INDEX index\_name

on table\_name (column\_name);

### Composite Indexes

CREATE INDEX index\_name

on table\_name (column1, column2);

## DROP INDEX Command

DROP INDEX index\_name;

## Show all INDEXES Command

EXEC sys.sp\_helpindex @objname = N'TABLE\_NAME';

# **SQL - CURSORS**

* A Cursor is a temporary memory that is allocated by the database server at the time of performing the Data Manipulation Language operations such as INSERT, UPDATE and DELETE etc., on a table.
* It holds the multiple rows returned by the SQL statement. You can use it to retrieve and manipulate data stored in the SQL tables.

Following is the diagram of the SQL server cursor life cycle −

* **Declare** keyword, we need to specify the name and the data type of the Cursor after the Declare keyword along with the SELECT statement.
* **Open** keyword, Once we declare the cursor we need to open it to store and retrieve data from the result set,
* **Fetch** keyword , we retrieve the rows using the **Fetch** keyword.
* **Close** the cursor ,Once we perform desired operations on the retrieved data we need to close the cursor.
* **Deallocate,** where we erase the definition of the cursor.

**Example :**

begin

declare @empId int;

declare @firstName varchar(15);

declare @salary decimal(6,2);

declare empcur cursor for

select EmpCode,EmpFName,Salary from EMPLOYEE where DEPTCODE=20;

open empcur;

fetch next from empcur into @empId,@firstName,@salary;

while @@FETCH\_STATUS=0

begin

print cast(@empId as varchar(15))+ ' ' +@firstName+' '+cast(@salary as varchar(15));

fetch next from empcur into @empId, @firstName,@salary;

end

close empcur;

deallocate empcur;

end;

# **SQL - TRIGGERS**

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
* A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
* A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

**Example :**

**Creating Trigger**

Create trigger EmpSalCheck on employee for update as

begin

declare @oldsal decimal(6,2);

declare @newsal decimal(6,2);

select @oldsal = salary from deleted;

select @newsal = salary from inserted;

if ( @oldsal > @newsal )

begin

print 'New salary cannot be less than old salary';

Rollback;

end;

else

begin

print 'New salary is updated';

commit;

end;

end;

**Implicitly trigger of trigger when that particular even is called.**

update EMPLOYEE set salary = salary-1;

# **T-SQL - 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:**

Create procedure <procedure\_Name>

As

Begin

<SQL Statement>

End

Go;

**Execute a Stored Procedure:**

EXEC procedure\_name;

**Example:**

CREATE PROCEDURE SelectCustomerstabledata

AS

SELECT \* FROM Testdb.Customers

GO;

EXEC SelectCustomerstabledata;

## Stored Procedure With One Parameter

CREATE PROCEDURE <procedure\_Name> <Variabel\_decleration> <Data\_type>  
AS  
<SQL Statement>   
GO;

EXEC <procedure\_Name> <Variabel\_decleration> =Value;

## Stored Procedure With Multiple Parameters

CREATE PROCEDURE <procedure\_Name>

<Variabel\_decleration> <Data\_type>,<Variabel\_decleration> <Data\_type>  
AS  
<SQL Statement>   
GO;

EXEC <procedure\_Name>

<Variabel\_decleration> =Value, <Variabel\_decleration> =Value;

Example :

create procedure EmpSalaryIncrement @deptId as int as

begin

declare @empId int;

declare @firstName varchar(15);

declare @salary decimal(6,2);

declare empcur cursor for

select EmpCode,EmpFName,Salary from EMPLOYEE where DEPTCODE=@deptID;

open empcur;

fetch next from empcur into @empId,@firstName,@salary;

while @@FETCH\_STATUS=0

begin

if @salary >=3000

begin

set @salary = @salary + 1000;

end

else

begin

set @salary = @salary +500;

end;

update EMPLOYEE set salary = @salary where EmpCode = @empId;

print cast(@empId as varchar(15))+ ' ' +@firstName+' '+cast(@salary as varchar(15));

fetch next from empcur into @empId, @firstName,@salary;

end;

close empcur;

deallocate empcur;

end;

execute EmpSalaryIncrement 10

select \* from EMPLOYEE

begin transaction

commit

**T-SQL - STRING FUNCTIONS**

SQL string functions are used primarily for string manipulation. The following table details the important **string functions –**

[ASCII()](https://www.tutorialspoint.com/sql/sql-string-functions-ascii.htm) Returns numeric value of left-most character

Ex : Select ASCII (‘word’)

[CHAR()](https://www.tutorialspoint.com/sql/sql-string-functions-char.htm) Returns the character for each integer passed

EX: Select CHAR(97);

N[CHAR()](https://www.tutorialspoint.com/sql/sql-string-functions-charindex.htm) Unicode character will come as output for a given integer.

Ex: Select NCHAR(300)

[CHARINDEX()](https://www.tutorialspoint.com/sql/sql-string-functions-charindex.htm) Returns the position of a substring within the given string.

Ex: Select CHARINDEX('G', 'KING')

[LEFT()](https://www.tutorialspoint.com/sql/sql-string-functions-left.htm) Returns the extracting string.

EX: Select LEFT('WORLD', 4)

[RIGHT()](https://www.tutorialspoint.com/sql/sql-string-functions-right.htm) Returns the rightmost characters from the actual(current) string.

Ex: Select RIGHT('INDIA', 3);

[SUBSTRING()](https://www.tutorialspoint.com/sql/sql-string-functions-substring.htm) Returns the part of the character.

Ex: Select SUBSTRING ('WORLD', 1,3);

[LEN()](https://www.tutorialspoint.com/sql/sql-string-functions-len.htm) Returns the length of the given string.

Ex: Select LEN('HELLO');

[LOWER()](https://www.tutorialspoint.com/sql/sql-string-functions-lower.htm) Returns returns the lower case of the given string.

Ex: Select LOWER('SQLServer');

[UPPER()](https://www.tutorialspoint.com/sql/sql-string-functions-upper.htm) returns a string that has all the characters in upper case.

Ex: Select UPPER('SqlServer');

[LTRIM()](https://www.tutorialspoint.com/sql/sql-string-functions-ltrim.htm) Returns a string after removing all the white spaces and characters from the string found on the left side.

EX: Select LTRIM(' WORLD');

[RTRIM()](https://www.tutorialspoint.com/sql/sql-string-functions-rtrim.htm) Returns a string after removing all trailing blanks.

Ex: Select RTRIM('INDIA ') ;

[REPLACE()](https://www.tutorialspoint.com/sql/sql-string-functions-replace.htm) Returns a new string by replacing all the occurrences of the particular part of a string (substring) with a specified string.

Ex: Select REPLACE('INDIA', 'I', 'K');

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

Ex : Select REPLICATE('WORLD', 2);

[REVERSE()](https://www.tutorialspoint.com/sql/sql-string-functions-reverse.htm) Returns a reversed string.

Ex: Select REVERSE('WORLD');

[SOUNDEX()](https://www.tutorialspoint.com/sql/sql-string-functions-soundex.htm) Returns the Soundex string.

Ex: Select SOUNDEX('Smith'), SOUNDEX('Smyth');

[DIFFERENCE()](https://www.tutorialspoint.com/sql/sql-string-functions-difference.htm) returns an integer value measuring the difference between the SOUNDEX() values of two different expressions(strings).

Ex: Select Difference('Smith','Smyth');

[SPACE()](https://www.tutorialspoint.com/sql/sql-string-functions-space.htm) returns a string consisting of N number of space characters.

Ex: Select 'I'+space(1)+'LOVE'+space(1)+'INDIA';

[STUFF()](https://www.tutorialspoint.com/sql/sql-string-functions-stuff.htm) Returns a new string by inserting the second expression at the specified deleted place.

EX: Select STUFF('ABCDEFGH', 2,4,'IJK')

[STR()](https://www.tutorialspoint.com/sql/sql-string-functions-str.htm) Returns a number as string.

Ex : Select STR(187.369,6,2);

[UNICODE()](https://www.tutorialspoint.com/sql/sql-string-functions-unicode.htm) Returns an integer value of the first character.

Ex : Select UNICODE('RAMA');

[PATINDEX()](https://www.tutorialspoint.com/sql/sql-string-functions-patindex.htm) Returns the position of a pattern in a string.

Ex : Select PATINDEX('I%','INDIA');

[FORMAT()](https://www.tutorialspoint.com/sql/sql-string-functions-format.htm) Returns the formatted string.

Ex : SELECT FORMAT ( getdate(), 'D');

[CONCAT()](https://www.tutorialspoint.com/sql/sql-string-functions-concat.htm) Returns concatenated string

Ex : Select CONCAT('A',',','B',',','C') ;

[CONCAT\_WS()](https://www.tutorialspoint.com/sql/sql-string-functions-concat-ws.htm) Returns concatenate with separator

[ESCAPE()](https://www.tutorialspoint.com/sql/sql-string-functions-escape.htm) Returns a text with escaped characters.

[QUOTENAME()](https://www.tutorialspoint.com/sql/sql-string-functions-quotename.htm) Returns a string with a delimiter

[STRING\_AGG()](https://www.tutorialspoint.com/sql/sql-string-functions-string-agg.htm) Concatenates the values of string expressions and places separator values between them.

[STRING\_SPLIT()](https://www.tutorialspoint.com/sql/sql-string-functions-string-split.htm) Splits a string into rows of substrings.

[TRANSLATE()](https://www.tutorialspoint.com/sql/sql-string-functions-translate.htm) Returns a string from the first argument.

[TRIM()](https://www.tutorialspoint.com/sql/sql-string-functions-trim.htm) Returns a trimmed string.

# **T-SQL - DATE FUNCTIONS**

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

GETDATE() It will return the current date along with time.

Syntax: GETDATE()

Ex : Select getdate() as currentdatetime;

DATEPART() It will return the part of date or time.

Syntax : DATEPART(datepart, datecolumnname);

EX : Select datepart(day, getdate()) as currentdate;

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

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

Syntax : DATEADD(datepart, number, datecolumnname);

Ex : Select dateadd(day, 10, getdate()) as after10daysdatetimefromcurrentdatetime;

DATEDIFF() It will display the date and time between two dates.

Syntax: DATEDIFF(datepart, startdate, enddate);

Ex : Select datediff(hour, 2015-11-16, 2015-11-11) as

differencehoursbetween20151116and20151111;

CONVERT() It will display the date and time in different formats.

Syntax : CONVERT(datatype, expression, style);

Ex : SELECT CONVERT(VARCHAR(19),GETDATE())

SELECT CONVERT(VARCHAR(10),GETDATE(),10)

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

# **T-SQL - 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)