



<b>INDEX</b>		
<b>S.No</b>	<b>Topic</b>	<b>Page No</b>
<b>SQL</b>		
1	Introduction to SQL	3
2	SQL Commands	9
3	Data types	10
4	Creating Database & Table	11
5	DDL, DML & DRL Commands	13
6	Operators	22
7	Different Clauses	30
8	Constraints	37
9	Functions	53
10	Joins	62
11	Normalization of Data	78
12	Indexes	84
13	Set Operators	89
14	Sub Queries	93
15	Synonyms	100
16	Views	102
<b>T-SQL</b>		
17	Introduction T-SQL	106
18	Basic Programs	108
19	Stored Procedures	113
20	Functions	124
21	TCL & DCL Commands	133
22	Cursors	139
23	Triggers	150
24	Exception Handling	175
25	Transforming of Data (Backup)	184
26	Question & Answers	189

## Introduction

### What is Front end application?

The application where user will interact is called frontend application.

These are developed by using frontend tools like .Net, Java, PHP, Etc..

EG:- Facebook. Gmail...

### What is Back end application?

The application where exactly user data will be maintained is called Backend application.

Backend application will be developed by using Backend tools like MS SQL Server, MY SQL, Excel, D2K,.Etc..

MS SQL Server is a product of -Microsoftl.

### What is Database?

A Collection of meaningful/ processed / organized data is called as Database. A database is a place to store data.

The main purpose of database is to operate large amount of information by storing, retrieving and managing.

There are many databases available like MySQL, Sybase, Oracle, Mango DB, Informix, SQL Server etc.

Eg:-

1) 111	Rama	25000	27
--------	------	-------	----

The above data is meaning less data

2) Rama Salary was 25000

3) Rama age is 27

The above data was meaningful / Processed data

In any database we will work on Data, it will occupy less memory.

## What is SQL

- SQL stands for **Structured Query Language**.
- SQL is just a query language, it is not a database. To perform SQL queries, you need to install any database.
- It is designed for managing data in a relational database management system (RDBMS).
- It is pronounced as S-Q-L or sometime See-Qwell.

## Why SQL is required

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database
- To retrieve data from a database

## What is RDBMS

**RDBMS** stands for Relational Database Management Systems.

All modern database management systems like MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

## Difference between DBMS and RDBMS

DBMS	RDBMS
DBMS applications store <i>data as file</i> .	RDBMS applications store <i>data in a tabular form</i> .
DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
DBMS uses file system to store data, so there will be <i>no relation between the tables</i> .	In RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.
DBMS <i>does not support distributed database</i> .	RDBMS <i>supports distributed database</i> .
DBMS is meant to be for small organization and <i>deal with small data</i> . it supports <i>single user</i> .	RDBMS is designed to handle large amount of data. it supports multiple users.
Examples of DBMS are file systems, <i>xml</i> etc.	Example of RDBMS are <i>mysql, postgre, sql server, oracle</i> etc.

## Degrees of Relations ships in RDBMS

- 1) One to one relation
- 2) One to many relation
- 3) Many to many relation

One to one relation:-

A row of a table is associated with a row in another table called one to one relation.

One to many relation:-

A row of a table is associated with multiple rows in another table is called one to many relation.

Many to many relation:-

Many rows in a table is associated with many rows in another table is called as Many to many relation.

## What is table

The RDBMS database uses tables(Object) to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

## What is field

Field is a smaller entity of the table which contains specific information about every record in the table.

## What is row or record

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table.

## What is column

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in that column which contains all information about name column.

## NULL Values

The NULL value of the table specifies that the field has been left blank during record creation. It is totally different from the value filled with zero or a field that contains space. By default every column will accept the NULL values.

## SQL Syntax

**SQL** is not case sensitive. Generally SQL keywords are written in uppercase.

SQL statements are dependent on text lines. We can place a single SQL statement on one or multiple text lines.

SQL statements are started with any of the SQL commands/keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the statement ends with a semicolon (;).

Example of SQL statement:

**SELECT** "column\_name" **FROM** "table\_name";

Why semicolon is used after SQL statements

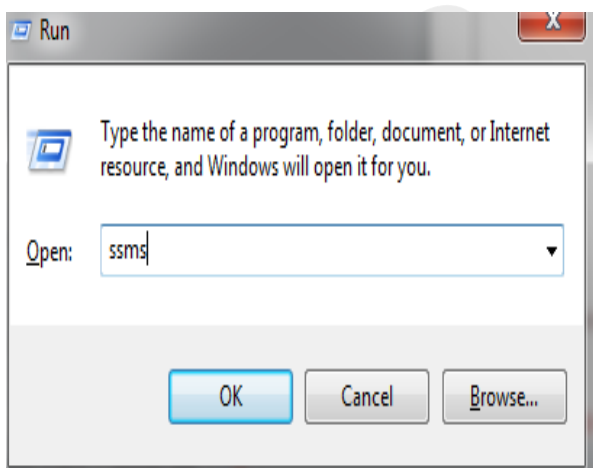
Semicolon is used to separate SQL statements. It is a standard way to separate SQL statements in a database system in which more than one SQL statements are used in the same call.

## Steps to open MS SQL Server

Start → Programs → MS SQL server 2008 / 12/ 14 → SQL Server Management Studio

(OR)

Go to Run (Windows +R) type -**ssms** then OK it will open SQL Server home page



Server Type:-

Server type= Database Engine

This will use to store the large amount of data processing the data and providing security. Here data will store I 2D format (Rows& columns).

By using SQL Server we can develop

- SQL Server Reporting Services [SSRS]
- SQL Server Analysis Services [SSAS]
- SQL Server Integration Services[SSIS]

Server Name:-

It is used to set the server name where SQL server was installed.

Authentication:-

It is the process of checking credentials means Username & Password

We can connect to SQL Server database in 2 ways

- By using Windows Authentication
- By using SQL Server Authentication

When we connect to SQL Server through Windows authentication no need to give Credentials.

Default user name is -**sa** (System Administrator)



## SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Retrieval Language (DRL)
- Transaction Control Language (TCL)
- Data Control Language (DCL)

Command	Command	Description
<b>DDL</b>	CREATE	Creates a new table, a view of a table, or other object in database
	ALTER	Modifies an existing database object, such as a table.
	DROP	Deletes an entire table, a view of a table or other object in the database.
<b>DML</b>	INSERT	Creates a record
	UPDATE	Modifies records
	DELETE	Deletes records in the table
<b>DRL</b>	SELECT	Retrieves certain records from one or more tables
<b>TCL</b>	commit	It commits the transaction
	rollback	Cancelling the transaction to particular point (Save point)
	savepoint	Save the transaction gives a name
<b>DCL</b>	GRANT	Gives a privilege to user
	REVOKE	Takes back privileges granted from user

## SQL Data Types

The SQL data type defines a kind of value that a column can contain.

In a database table, every column is required to have a name and a data type.

Data-type	Syntax	Explanation
Integer	INTEGER	The integer data type is used to specify an integer value.
Character	Char	To store the single (or) group of characters, Size of Char is fixed. Max size:8000 characters
Varchar	Varchar	To store the single (or) group of characters, Size of Char is not fixed.
Text	Text	Similar to varchar(max)
Nchar	nchar	The size of nchar is fixed like char. By using this we can store National Character data i.e if we want to store Chinese, Japanese, French are any other different language then we go for nchar Max size:8000 characters
Real	REAL	The real integer is used to specify a single precision floating point number.
Decimal	DECIMAL(P,S)	It specifies a decimal value. Here 'p' is precision value and 's' is scale value.
Double precision	DOUBLE PRECISION	It specifies double precision floating point number.
Float	FLOAT(P)	It specifies floating-point value e.g. 12.3, 4.5 etc. Here, 'p' is precision value.
Date	DATE	It stores year, month and days values.
Time	TIME	It stores hour, minute and second values
Money	MONEY	It stores the money related values, like salary.

## SQL Create Database

Databases are basically divided into 2 types.

1. System defined database
2. User defined database

System Defined Database:-

These databases are installed automatically when we install SQL Server Management Studio.

Different types of system defined databases are

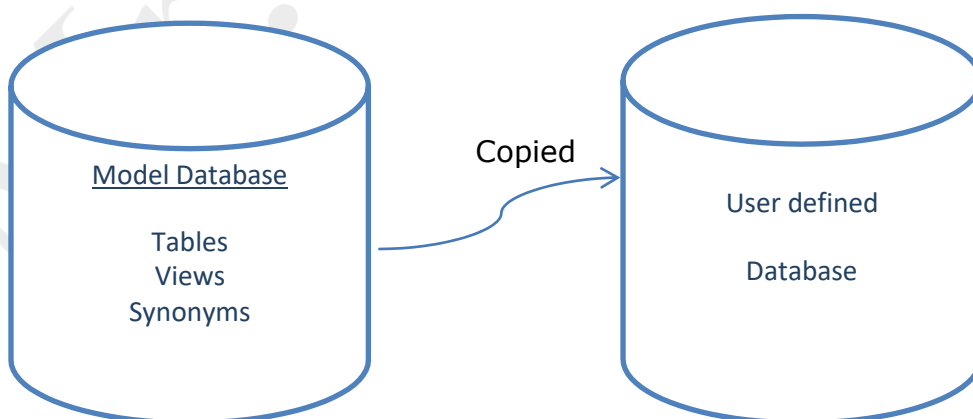
1. Master
2. Model
3. Temp db
4. MSdb

Master:-

- It consists of configuration settings that are required to run the SQL Server Management studio.
- It is used to maintain the user credentials & Authentication

Model:-

- Model database is a Template for all the other databases
- Whenever we are creating user defined database then the contents of model database are copied into user defined database.



Temp db:-

- It is used to perform temporary calculations and operations.

MS db:-

- It is used to schedule jobs & alerts

->To view all database details in system

Select \* from sysdatabases;

-> To view all system user details

Select \* from sysusers;

-> To view all login details..

Select \* from syslogins;

-> To view all object details in the system

Select \* from sysobjects;

User defined database:-

The database that was created depending on the user requirement is called user defined database.

Eg: - College, Employee, School...

### **Note:-**

Once we create a new database in is SQL server then the system will create 2 types of datafiles for each and every database in sql server.

Those are

1) Primary Data file(.mdf file):-

It will store all database tables data with an extension of **.mdf** (Master Data File).

2) Log data file (.ldf file):-

It will store transaction or overall query information which was executed by the user on database objects (Tables, Synonyms, Views, etc...) will save with an extension of **.ldf**(log data file)

The above two files are used to transfer the database information one system other system / location.

## **DDL Commands**

- These commands are used for create database(or) database objects like Table, Views, Synonyms, stored procedures , etc...
- By using DDL we can modify the database objects like adding a new column, removing a column, change the data type name etc..
- By using DDL commands we can remove database (or) Tables.

### Creating Database (User defined)

// Syntax for creating a database

Syntax

*CREATE DATABASE database\_name;*

Eg:- create database Companydb;

Select query and press 'F5' to execute.

- If you want to add tables in that database, you can use CREATE TABLE statement.
- Always database name should be unique within the RDBMS.

// Write a query to view the information of the database.

Syntax

*SP\_helpdb databasename;*

Eg:-sp\_helpdb Companydb;

- When we execute the above query it will display entire information of database like database name, owner name, size, creation time etc,.....
- SP\_helpdb is predefined stored procedure which was written by Microsoft.

// Write a query to use database.

Syntax

*Use databasename;*

*Eg:-use Companydb;*

// Write a query to rename the database.

Syntax

*SP\_renamedb old\_databasename, new\_databasename;*

*Eg:-sp\_renamedb Companydb,Company;*

// Write a query to remove the database

- If you want to delete or drop an existing database in a SQL schema, you can use SQL DROP DATABASE

Syntax

*Drop database databasename;*

*Eg:-drop database Company;*

### Creating Table (User defined)

// Syntax for creating a table

Syntax

Create table "tablename"("column1" "data type",  
"column2" "data type", -column3" "data type",..  
"columnN" "data type");

Eg:-

Create table emp (empid int,empname varchar(10), designation  
varchar(10),salary money, doj date);

// Write a query to view the information of the table.

Syntax

*SP\_help tablename;*

*Eg:-sp\_help emp;*

Points to remember at the time of Table creation:-

- A table name should be unique within the same database.
- A column name should be unique in the same table definition.
- A table name should not start with numbers, special characters except ( ) underscore symbol.
- Don't provide space in the table name, if required can be used underscore only.
- Don't use the reserved keywords like Insert, Delete, Update and Select as a table name.
- A table name should contain minimum of 1 character and maximum 128 characters.
- A table should contain min 1 column and max 1024 columns.
- A table contains unlimited rows.

Alter Command:-

- This command is used to do modifications for database objects like tables, views, synonyms, stored procedures, triggers.....
- By using alter command we can add a new column for existing table as well as we can remove a column from the table.
- By using alter command we change the data type name of column.
- We can change the size of the data type for column(s).

// Write a query to add a new column in emp table with name Gender.

Syntax

*Alter table tablename add Columnnamedatatype;*

*Eg:-Alter table emp add Gender char(1);*

Note: - we can add a new column even if the table consists of data.

// Write a query to change datatype name of empid column from int to bigint.

Syntax

*Alter table tablename alter column Columnname new datatype;*

*Eg:-Alter table emp alter column empid bigint;*

// Write a query to change the size of datatype from varchar(10) to varchar(50) for empname .

Syntax

*Alter table tablename alter column Columnname new datatype size;*

*Eg:-Alter table emp alter column empname varchar(50);*

Truncate Command:-

- This command is used to remove the data from the table

Syntax

*Truncate table tablename;*

*Eg:-Truncate table emp;*

- Before we are using this command table should have some records to perform this command.



### Drop Command:-

- This command is used to remove the data from the table along with table structure.

#### Syntax

*Drop table tablename;*

Eg:-Drop table emp;

// Write a query to remove doj column from emp table.

#### Syntax

*Alter table tablename drop column coumnnname;*

Eg:- Alter table emp drop column doj;

// Write a query to rename the table.

#### Syntax

*Sp\_rename <tablename>,<New\_table\_name>;*

Eg:- sp\_rename emp , emp\_tbl;

// Write a query to rename the column name in table.

#### Syntax

*Sp\_rename 'tablename.oldcolumnname', 'Newcolumnname;*

Eg:- sp\_rename 'emp-tbl.designation','cader';

// Write a query to copy the table in new table.

#### Syntax

*Select \* into <destination\_table> FROM <source\_table>*

Eg:-select \* Into emp\_tbl\_new From emp\_tbl;

- It will copy the entire table with data in another new table.

## **DML Commands**

- These commands are used to Insert (or) Delete (or) Update the records in the Table.

### **Insert:-**

It is used to insert the record into the table.

Syntax (Implicit method)

*Insert into <tablename> values (Val1,Val2, Val3...);*

Eg:-

Insert into emp\_tbl values (111,'Rama','CEO',25000,'M');

Note:-

The above insert query is used to insert all the records in the table. If you want to insert specific records in the table we have to use

Syntax (Explicit method)

*Insert into <tablename> (Colname)values(value of column);*

Eg:-

Insert into emp\_tbl(empname,salary)values ('Basha',12000);

### **Inserting multiple records into the table:-**

Syntax

*Insert into <table Name>  
(col1,col2, col3) values (Col1value, Col2value, Col3value),;*

Eg:-

Insert into emp\_tbl(empid,empname,cader,salary,Gender) values  
(123,'lakshman','MD',18000,'M'), (121,'sita','HR',2000,'F'),  
(124,'Ghouse','GM',20000,'M');

-- with the aboue statement we inserted 3 records at a time.



**Identity (Seed, Increment)**

- This is a predefined function which will use to generate numeric values to particular column in a table.
- This function contains 2 arguments
  - Seed :- will represent starting / initial id value
  - Increment: - will represent incremental value between id's.
- The default value of seed & increment is (1,1)

Example:-

Create table Tbl\_Emp (Eid int identity , ename varchar(20), age int)

Insert into Tbl\_Emp values (1,'sai',29);// not allowed

Insert into Tbl\_Emp values ('sai',29);// allowed

**Note:-**

If the user want to insert the values into identity column by explicitly following syntax for that

Syntax:-

Set identity\_insert <Tbl\_Name> on/off;

E.g:-

Set identity\_insert Tbl\_Emp off;

**Delete:-**

It is used to delete specific records (or) group of records from table.

Use the WHERE clause to Delete only specific records.

*Differences between Truncate and Delete*

Truncate	Delete
Its DDL Command	Its DML Command
Using this we can't delete specific record	We can delete specific record from table
By using this we can delete all the records from table	We can delete all the records from the table
We can't roll back the deleted data	We can roll back the deleted data (restoring to table)
Where <b>condition</b> will not work	Where <b>condition</b> will work
Truncate will delete all the rows at a time	Delete will delete the rows on by one
It will work faster	It will work slowly

// Write a query to delete the table data (all records).

Syntax

*Delete from <table-name>;*

Eg:-Delete from emp\_tbl;

// Write a Delete query on condition.

Syntax

*Delete from <table-name> where <condition>;*

Eg:-

Delete from emp\_tbl where empid=123;

Delete from emp\_tbl where empname='Basha';

Delete from emp\_tbl where salary>=10000;

### **Update:-**

It is used to update the data which is available in the table.

- UPDATE can update one or more records in a table.
- Use the WHERE clause to UPDATE only specific records.

Syntax

*Update <table\_name> set <col1> = value1 where<condition>;*

Eg:-

Update emp\_tbl set salary=20000 where empid=121;

Update emp\_tbl set name=Hari where empid=161;

## **DRL Commands**

This command is used to display a specific records (or) group of records from Table.

- The SELECT statement retrieves data from a database.
- The data is returned in a table-like structure called a result-set.
- Select is the most frequently used action on a database.

Syntax

*Select\* from <table\_name>;*

Eg:-

Select \* from emp\_tbl;

Select empid , empname, salary from emp\_tbl;

Select \* from emp\_tbl where empid=121;

## Operators

- Operators are used to perform operation on two (or) more operands.
- Operators are classified into 5 types
  - Arithmetical Operators
  - Logical operators
  - Comparison operators
  - Range operators
  - String Operators

### *Arithmetical Operators:-*

These operators are used to perform arithmetical operation's like Addition, Div, Sub, Mul, Division...

Operators	Descriptions
+	It is used to add containing values of both operands
-	It subtracts right hand operand from left hand operand
*	It multiply both operand's values
/	It divides left hand operand by right hand operand and returns Quotient
%	It divides left hand operand by right hand operand and returns Reminder

What is an Expression?

- An expression is the combination of 2 (or) more operators.
- Whenever we are evaluating an expression operation we have to evaluate based on the priority of the operation.

Eg:-  $23+4-5*2+7*2-4/2$

1<sup>st</sup> Priority → \*,/,%

2<sup>nd</sup> Priority → +,-

3<sup>rd</sup> Priority → =

Note:- if numerator < zero the quotient =0 & numerator = reminder.

Eg 1:- Create a table with name student with columns stno, stname, sub1, sub2, sub3.

Create table student

```
(  
    stno int, stname varchar(15), sub1 int, sub2 int, sub3 int  
);
```

Insert into student values (002,'Aashrith',80,96,70);

*// WAQ to display student details.*

Select \* from student;

*// WAQ to display student details along with total marks.*

Select \*, (sub1+sub2+sub3) from student ;

o/p

stno	stname	sub1	sub2	sub3	(No column name)
1	Aashrith	80	96	70	246

Select \*,(sub1+sub2+sub3) as 'Total' from student ;

O/P ( create a column as Total name)

stno	stname	sub1	sub2	sub3	Total
1	Aashrith	80	96	70	246

*// WAQ to display details along with Total marks , % of marks.*

Select \*,(sub1+sub2+sub3) as 'Total',(sub1+sub2+sub3)/3 as 'Percentage' from student ;

O/P

Stno	stname	sub1	sub2	sub3	Total	Percentage
1	Aashrith	80	96	70	246	82



Eg 2:- Create a table with name Employee with columns EID, Ename, basicsal.

Create table Employee( EID int, Ename varchar(10), BasicSal money);

// WAQ to display Employee details along with DA, HRA, Gross.

DA= 0.4\*BasicSal.

HRA=0.6\*BasicSal.

Gross=BasicSal+DA+HRA.

Select\*,(0.4\*Basicsal) as 'DA',(0.6\*BasicSal) as 'HRA' ,  
(BasicSal+(0.4\*BasicSal) + (0.6\*BasicSal)) as 'Gross' from Employee;

O/P

EID	Ename	BasicSal	DA	HRA	Gross
189	Sekhar	20000.	8000	12000.	40000.

// WAQ to display total Salary of Sekhar.

Select \*,(BasicSal+(0.4\*BasicSal) + (0.6\*BasicSal)) as 'Total' from  
Employee where Ename='Raja';

### To Rename a Colun in the Table:-

Syntax:-

Sp\_Rename \_Table\_Name.Column\_Name`,`New\_Column`,`\_Column`;

### Logical operators

There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, you can use logical operators in the WHERE clause, which allows you to combine more than one condition.

S.No	Logical Operators	Description
1	OR	For the row to be selected at least one of the conditions must be true.
2	AND	For a row to be selected all the specified conditions must be true.
3	NOT	For a row to be selected the specified condition must be false.

**Note:-Create a table-"Student\_Tbl" to perform logicaloperators on that table.**

#### 1. "OR"

If you want to select rows that satisfy at least one of the given conditions, you can use the logical operator, \_OR`.

Eg: - If you want to find the names of students who are studying either Maths or Science from the table Student\_tbl, the query would be like,

#### Query:-

```
SELECT first_name, last_name, subject FROM student_tbl
WHERE subject = 'Maths' OR subject = 'Science' ;
```

Column1 Satisfied?	Column2 Satisfied?	Row Selected
YES	YES	YES
YES	NO	YES
NO	YES	YES
NO	NO	NO

## 2. "AND"

If you want to select rows that must satisfy all the given conditions, you can use the logical operator, `_AND'`.

Eg:- To find the names of the students between the age 10 to 15 years from the table `Student_tbl`, the query would be like:

Query:-

```
Select first_name, last_name, age from Student_tbl  
where age >= 10 AND age <= 15;
```

Column1 Satisfied?	Column2 Satisfied?	Row Selected
YES	YES	YES
YES	NO	NO
NO	YES	NO
NO	NO	NO

## 3. "NOT"

If you want to find rows that do not satisfy a condition, you can use the logical operator `_NOT'`. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned.

EG: - If you want to find out the names of the students who do not play football, the query would be like:

Query:-

```
Select first_name, last_name, games from Student_Tbl  
where NOT games = 'Football';
```

### Comparison Operators:

Comparison operators are used to compare the column data with specific values in a condition. And also used along with the SELECT statement to filter data based on specific conditions.

Comparison Operators	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

### Queries:-

- 1) Select \* from *Student\_Tbl* where Fee>15000;
- 2) Select \* from *Student\_Tbl* where Fee<8000;
- 3) Select \* from *Student\_Tbl* where Fee=10000;
- 4) Select \* from *Student\_Tbl* where Fee != 18000;
- 5) Select \* from *Student\_Tbl* where Fee <> 18000;

### Range Operators:

These operators which selects data according to particular range.

2types of Range operators are there,

- 1) Between
- 2) Not between

**Between:-**

This will select the data between any ranges. The values can be numbers, text, or dates.

**Example Query:-**

*1) Select \* from Student\_Tbl where Fee between 10000 and 12000;*

With this query it will select data between 10000 and 12000.

**Not between:-**

To display the data outside the range of the previous example, use not between.

**Example Query:-**

*Select \* from Student\_Tbl where Fee not between 10000 and 12000;*

**String operators:-**

These operators are used in where clause to search the data by using a specific pattern. These are two types,

1. Like (Like keyword allows you to select records that match with pattern)
2. Not like (Not keyword allows you to select records that do Not match the pattern.)

**Eg:-**

S% → String starts with 'S'.

%S → String ends with 'S'.

%S% → in between 'S'.

**Example Queries:-**

*1) Select \* from Student\_Tbl where sname like 'S%';*

It will display all the names starts with 'S' letter.

*2) Select \* from Student\_Tbl where sname like 'S%' and Fee > 12000;*

It will display all the names starts with 'S' and Who's Fee is greater than 12000.

3) *Select \* from Student\_Tbl where sname like 'S%S';*

It will display all the names starts with 'S' and end's with 'S'.

4) *SELECT \* FROM Student\_Tbl WHERE Country NOT LIKE '%Ind%';*

The above SQL statement selects all students with Country NOT containing the pattern "Ind".

### **Different Clauses**

By using clauses we can provide some additional facilities to the query which we are processing like filtering, Searching, fetching the records in a particular table.

SQL Server supports several Clauses like

- ✓ Where
- ✓ Order by
- ✓ Top n
- ✓ Group by
- ✓ Having

#### **Where:-**

- The SQL '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 then only it returns specific value from the table. You would use Where clause to filter the records and fetching only necessary records.
- The Where clause is not only used in Select statement, but it is also used in **Update, Delete statement**, etc.
- WHERE is followed by a condition that returns either true or false.

Syntax:-

**Select <Column\_ List>FROM <Table\_Name>Where [Condition];**

E.g:-

->select eid, name, dept from tbl\_emp where address ='Chennai';

It will return the records from tbl\_Emp whose address is Chennai.

->update tbl\_emp set salary=35000 where eid=1;

It will update the salary to 35000 whose eid=1 in Tbl\_Emp table.

->delete from tbl\_emp where eid=7;

It will delete employee details whose eid=7 in Tbl\_Emp table.

**Order by:-**

- It will arrange the values either Ascending or Descending order.
- By default order by clause will arrange the column values in Ascending order.
- Order by allows sorting by one or more columns.
- If we want to arrange the data in descending order then we will use -Desc keyword.
- Order by clause can be implemented in select statement only.
- This will sort Numeric, Character type values. It's a temporary arrangement, the results returned by orderby clause is for viewing purpose only and it can't be sorted as such in the database.
- Order by clause should be last in the 'select' statement.

Syntax:-

**Select <Column\_List> from <Table\_Name> where <Condition>  
Order by <Column1, Column2...> [ASC] [DESC];**

E.g:-

->select \* from tbl\_emp order by Name;

It will display the employee names in Ascending order.

->select \* from tbl\_emp order by gender desc, Salary asc;

In the above query 1<sup>st</sup> it will arrange gender column in descending order after that based on gender it will arrange Salary column in ascending order.

Note:-

Based on condition we will use where clause.

**Top n :-**

- The SQL Top clause is used to fetch Top N number records from a table.
- 'n' specifies no of records.
- Select Top is useful when working with very large datasets.
- Non SQL Server databases use keywords like LIMIT, OFFSET, and ROWNUM.

Syntax:-

Select Top n <Column\_Name> From <Table\_Name>;

E.g:-

-> select top 3 eid, name, salary, Dept from Tbl\_Emp;

It will return the top 3 records eid, name, salary, Dept data from the given table.



**Group by:-**

- The SQL '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.
- Group By returns one records for each group. We can group by one or more columns.
- Group By typically also involves aggregates: COUNT, MAX, SUM, AVG, etc.

**Syntax:-**

**SELECT <Column\_Names> from <Table\_Name> where [condition]  
Group By <Column\_Names>;**

E.g:-

->select Dept, sum (salary) from Tbl\_Emp group by dept;  
It will return group wise department sum of salary

->select Dept, sum (salary) from Tbl\_Emp group by dept order by Dept desc;  
It will return group wise department sum of salary in descending order.

->select address, count (\*) from Tbl\_Emp Where address ='chennai'  
group by address;  
It will return the count of employees whose address is 'Chennai'.

->Select count (\*) from Emp\_tbl inner join Dept\_Tbl on Emp\_tbl.Eid = Dept\_Tbl.id where Emp\_tbl.Salary> 20000 group by Dept\_ID order by Dept\_ID desc;  
--By using joins within 2 tables.

**Having:-**

- Having also used for filtering the data just like where clause.
- Having applies to summarized group records, whereas Where applies to individual records.
- Having requires that a Group by clause is present. Where and Having can be in the same query.
- The **Having** clause must follow the Group by clause in a query and must also precedes the Order by clause if used.

**Syntax:-**

**Select <Column\_Names> from <Table\_Name> where  
[condition] Group By <Column\_Names> Having [condition];**

-> Select dept, sum (salary) from Tbl\_Emp group by dept having sum (salary)>100000;

->Select count (Eid) as Count\_eid, address from Tbl\_Emp  
Where Address<>'Hyderabad' group by Address having count  
(Eid)>= 0 order bycount (Eid) desc;

**O/P:**

Count_eid	address
2	Chennai
1	Lucknow
1	Mumbai
1	Ongole

**Differences between Where and Having clauses**

Where	Having
Where clause cannot be used with aggregates.	Havingclause can be used with aggregates.
Where class is used for filtering individual rows.	Having clause is used for to filter groups.
Where comes before Group by, it filters rows before aggregate calculations are performed.	Having comes after Group by, it filters rows after aggregate calculations are performed.
Without __group by` we can use where clause.	Without __group by` we can't use having clause.

Having is slower than where clause and should be avoided when possible.

Where and Having can be used in a select query. In this case where clause is supplied first individual rows. The rows are then grouped and aggregate calculations are performed, and then the having clauses filter the groups.

**Roll up & Cube:-**

These are special clauses in SQL which are used to provide Sub-total & Grand total automatically.

These clauses can be implemented with Group by clause only.

If "n" is the number of columns listed in the ROLLUP, there will be n+ 1 level of subtotals.

**Syntax for Roll up:-**

**Select <Column"s > from <Table\_Name> group by (<Column"s>);**

**E.g:-**

Select Dept, count(\*) Count\_Num from Tbl\_Emp group by rollup(Dept);

Dept	Count_Num
DBA	1
HR	3
IT	2
<b>NULL</b>	<b>6</b>

Here NULL is the Grand Total.

Rollup clause key word will provide sub-total & Grand Total based on a single column only.

**CUBE:-**

If we want to find sub-total and grand total based on multiple columns then we will use `_Cube` clause keyword.

If "n" is the number of columns listed in the CUBE, there will be  $2^n$  subtotal combinations.

**Syntax for Cube:-**

**Select <Column's> from <Table\_Name> group by Cube (<Column1>,< column2>);**

**E.g:-**

Select Dept, count(\*) count\_Num, job from Tbl\_Emp group by cube(Dept,job);

Dept	count_Num	Job
HR	1	HR Admin
NULL	1	HR Admin
IT	2	SR. Developer
NULL	2	SR. Developer
HR	2	SR.HR Executive
NULL	2	SR.HR Executive
DBA	1	Sys Admin
NULL	1	Sys Admin
NULL	6	NULL
DBA	1	NULL
HR	3	NULL
IT	2	NULL

**Differences between Rollup & Cube**

<b>Rollup</b>	<b>Cube</b>
ROLLUP generates a result set that represents aggregates for a hierarchy of values in the selected columns.	CUBE generates a result set that represents aggregates for all combinations of values in the selected columns.
It's an extension to GROUP BY clause. It's used to extract statistical and summarized information from result sets. It creates groupings and then applies aggregation functions on them.	It's an additional switch to GROUP BY clause. It can be applied to all aggregation functions to return cross tabular result sets.
Produces only some possible subtotal combinations.	Produces all possible combinations of subtotals specified in GROUP BY clause and a Grand Total.

## **Constraints**

Constraints are used to restrict the insertion of unwanted data in any columns. We can create constraints on single or multiple columns of any table. Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table. It maintains the data integrity of the table. This ensures the accuracy and reliability of the data in the database. Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL Supports different type of Constraints they are

### *1) Not Null Constraint*

Indicates that a column cannot store NULL value

### *2) Default Constraint*

Provides a default value for a column when none is specified.

### *3) Unique Constraint*

Ensures that all values in a column are different.

### *4) Primary key Constraint*

A combination of a NOT NULL and UNIQUE. i.e. Uniquely identified each row/records in a database table.

### *5) Foreign key Constraint*

Ensure the referential integrity of the data in one table to match values in another table.

### *6) Check Constraint*

The CHECK constraint ensures that all values in a column satisfy certain conditions.

Not Null Constraint:-

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents unknown data.

Example:-

```
Create table customers(  
    id int not null,  
    name varchar (20)not null,  
    age int not null,  
    address varchar (25),  
    gender char (5),  
);
```

For Existing table:-

```
Alter table customers  
    Alter column address varchar(25)not null;
```

Default Constraint:-

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example:-

```
Create table customers(  
    id int not null,  
    name varchar (20)not null,  
    age int not null,  
    address varchar (25)default'Hyderbad',  
    gender char (5),  
);
```

Eg 1:- insert into customers2 (ID,NAME,AGE,GENDER) values (1,'ram',25,'M');

In the above insertion I didn't pass address but by default it will take as Hyderabad.

ID	NAME	AGE	ADDRESS	GENDER
1	ram	25	Hyderabad	M

Eg 2:-insert into customers2(ID,NAME,AGE,GENDER,ADDRESS)  
Values (2,'raghu',25,'M','Chennai');

In the above query it will take user value i.e 'Chennai' only it will skip the default constructor value.

Eg 3:- insert into customers2(ID,NAME,AGE,GENDER,ADDRESS)  
values(3,'Aashrith',25,'M',Null);

In this case also it didn't take 'Default value', because you specifically supply a value 'Null' so it will accept 'Null' only.

Note: - Whenever you miss the value / didn't supply the value in that case only it will take default value.

For Existing table:-

Syntax:-

```
Alter table <TBL_NAME>  
add constraint <CONSTRAINT_NAME>  
default <DEFAULT_VALUE>for <Existin_Column_Name>;
```

Example:-

Alter table customers2

Add constraint DF\_Cust\_Adrs

Default 'Hyderabad' for address;

Drop Default Constraint:-

Alter table <table\_name>Drop constraint<Constraint Name>;

Alter table customers2Drop constraintDF\_Cust\_Adrs;



*Unique Constraint:-*

- The UNIQUE Constraint prevents two records from having identical values in a particular column. The should not allow Duplicate values.
- A table can have only one Primary key. If you want to enforce on two or more columns then we use Unique key constraint.
- It will allow one Null value

*Example:-*

```
Create table Unique_Tbl  
(  
    id int not null,  
    name varchar(20),  
    EMail varchar(30)not null unique,  
    SSID char(10)not nullunique,  
    PANID char(10),  
    CellNO Int  
);
```

Table creates with two Unique key columns.

*Inserting Data:-*

Insert into Unique\_Tbl values

```
(1,'Aashrith','aashrith@mail.com',  
    'JUN102012','ASHJD',996669);
```

If you are trying to insert another record with same column values i.e Email & SSID it will through an error like below,

*Error Msg Like:-*

Violation of UNIQUE KEY constraint 'UQ\_\_Unique\_T\_\_A0D712A2567ED357'. Cannot insert duplicate key in object 'dbo.Unique\_Tbl'. The duplicate key value is (JUN102012 ).

The statement has been terminated.

Adding Unique key constraint by using alter method

Syntax:-

```
Alter table <Table_Name> Add constraint <Constraint_Name>  
unique<(Column_Name)>;
```

For multiple Columns:-

```
Alter table <Table_Name> Add constraint  
<Constraint_Name><(Column_Name1,Column_Name2)>;
```

Drop Unique Constraint:-

```
Alter table <Tbl_Name> drop constraint <ConstraintName>;
```

Note:-

- We cannot apply not null constraint on unique key column.
- We can apply unique key constraint on not null column.

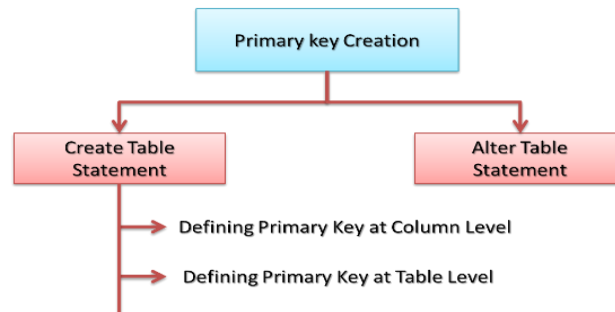
### **Primary key Constraint:-**

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL / Duplicate values. Whenever we are applying Primary key constraint on a column / table the automatically Not Null constraint will be applied. Primary key constraint will have name.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a 'composite key'. Composite primary key individual values are accepted duplicate values but duplicate combination should not be repeated. It can be defined on max 16 columns only. It is defined at end of the table definition.

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s). We can apply primary key constraint at the time of creating the table (or) after creating the table.

**NOTE:** If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).



Example 1:-

*Assigning Primary Key Constraint at the time of table creation*

```
Create table PK_Example  
(  
  No int primary key,  
  name varchar(15),  
  city varchar(15),  
  Cell bigint);
```

Inserting values:-

```
Insert into PK_Example values (1,'Sai','Hyderbad',9966640779);
```

if you try to insert the same number column `_No` i.e the value 1 it will through an error.

Error Message:-

Violation of PRIMARY KEY constraint 'PK\_\_PK\_Exxam\_\_3214D4A80425A276'. Cannot insert duplicate key in object 'dbo.PK\_Example'. The duplicate key value is (1).

The statement has been terminated.

Example 2:-

*Assigning Primary Key Constraint Existing table.*

```
Create table PK_Exp_Existing_tbl
(
  No int not null,
  name varchar(15),
  city varchar(15),
  Cell bigint not null
);
```

*The above table was existing table, now we want to add primary key constraint to the above table.*

Syntax:-

```
Alter table<table_name>add constarint<constarint_Name>primary
key <(column_name)>;
```

Query Eg:-

```
Alter table pk_exp_existing_tbl add constarintid_pk primary key
(id);
```

Example 3:-

For defining a Primary Key constraint on multiple columns.

Syntax:-

```
Create table PK_Exp_MultiCol_tbl
(
  No int not null,
  name varchar(15),
  city varchar(15),
  Cell bigint not null,
  Primary key(no,cell)
);
```

Example 4:-

For defining a Primary Key constraint on multiple columns on existing table.

**Synatx:-**

```
alter table<table_name>addconstraint<constraint_name>primary key
<(column1, column2)>;
```

query eg:-

```
alter table pk_exp_multicol_tbl2 add constraint pk_const2 primary key (no, cell);
```

Deleting Primary Key Constraint:-

Syntax:

Alter table<Table\_Name>drop constraint<Constraint\_Name>;

Eg:-

Alter table PK\_Exp\_MultiCol\_tbl2 drop constraint PK\_Const2;

**Note:-**Primary key always has a name. If you didn't specify it in 'create table' or 'alter table' statement, the key name is auto-generated.

### Foreign key Constraint:-

- It is used to establish the relationship between two (or) more tables.
- Referenced key in parent table must either be Primary Key or Unique Key.
- We can apply more than one foreign key constraint on single table, it's called **Composite Foreign Key**
- We cannot insert the record in foreign key column until unless the record is available in Primary key column.
- We cannot delete the record from primary key column until unless the record is deleted from foreign key column.
- It will accept null values.
- We cannot drop primary key constraint until the foreign key constraint is dropped.
- Foreign key involves two different tables. First one is **PARENTTABLE** or referenced Table and second one is **CHILD TABLE** or foreign table.
- Which table contains Primary key that's Parent table, Which table contains a foreign key constraint that's as child table.
- Column(s) of parent table column(s) of child table which are participating in foreign key should be of same data-type and size (column width).
- We can apply a foreign key constraint on table max 253 columns on a single table.
- By default foreign key constraint will allows us to enter duplicate & Null values.

## Defining Foreign Key Using Create table at Column Level

### For Example:-

To demonstrate this we will use two tables parent table with the name of Authors and child table with the name of Books Parent table authors is a simple table with 2 columns Author\_id and Author\_name. Where Author\_id is a Primary key column. You can add as many column as you want.

### Parent Table:-

```
createtableauthor
(
author_id int constraint athr_aid_pk primary key,
author_name varchar(30)
);
```

Now let's create our child table BOOKS. The structure of this table contain columns book\_id which will be the primary key for this table, book\_title and Book\_price and the 4th column will be book\_author\_id this column will be the foreign key which will reference the author\_id column of author table you can give whatever name to this column but data-type and the size (column width) of this column must be the same as of author\_id column in author table.

### Child Table:-

```
Create table books
(
book_id int,
book_title varchar(30),
book_price int,
book_author_id int constraint bok_ai_fk references author(author_id)
);
```

Defining foreign key using CREATE TABLE at table Level.

Create table books

```
(  
    book_id int,  
    book_title varchar(30),  
    book_price int,  
    book_author_id int,  
    constraint bok_ai_fk foreignkey (book_author_id) references  
    author(author_id)  
);
```

As you can see I defined the columns first and then I define foreign Key constraint in the last statement of Create table.

Define Foreign Key Using ALTER TABLE statement.

We define Foreign Key through ALTER TABLE statement when we already have a table and then we want to emphasize the constraint over that. In this scenario we will use ALTER TABLE statement.

Syntax:-

```
Alter Table    <Child_Table_Name>  Add constraint  
    <Constraint_name>  Foreign Key  
    < (Child_Tbl_Column)>  References  
    <Parent_Table_Name (Parent_Tbl_Column)>;
```

Eg:-

```
Alter table    books    add    constraint    bok_ai_fk    foreign key  
(book_author_id) references    author(author_id);
```

What is a foreign key with Cascade DELETE in SQL Server?

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in SQL Server.

A foreign key with cascade delete can be created using either a CREATE table statement or an ALTER table statement.

**Syntax:-**

Create Table <child\_table>

```
(  
    Column1 datatype [NULL | NOT NULL],  
    Column2 datatype [NULL | NOT NULL],  
    ...  
    Constraint fk_name  
        Foreign key (child_col1, child_col2, child_col_n)  
        References parent_table (parent_col1, parent_col2, parent_col_n)  
        On delete cascade  
        [ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]  
);
```

*child\_table*

The name of the child table that you wish to create.

*column1, column2*

The columns that you wish to create in the table. Each column must have a datatype. The column should either be defined as NULL or NOT NULL and if this value is left blank, the database assumes NULL as the default.

*fk\_name*

The name of the foreign key constraint that you wish to create.

*child\_col1, child\_col2, child\_col\_n*

The columns in child\_table that will reference a primary key in the parent\_table.

*parent\_table*



The name of the parent table whose primary key will be used in the child\_table.

parent\_col1, parent\_col2, ... parent\_col3

The columns that make up the primary key in the parent\_table. The foreign key will enforce a link between this data and the child\_col1, child\_col2, ... child\_col\_n columns in the child\_table.

#### *ON DELETE CASCADE*

It specifies that the child data is deleted when the parent data is deleted.

#### *ON UPDATE*

Optional. It specifies what to do with the child data when the parent data is updated. You have the options of NO ACTION, CASCADE, SET NULL, or SET DEFAULT.

#### *NO ACTION*

It is used in conjunction with ON DELETE or ON UPDATE. It means that no action is performed with the child data when the parent data is deleted or updated.

#### *CASCADE*

It is used in conjunction with ON DELETE or ON UPDATE. It means that the child data is either deleted or updated when the parent data is deleted or updated.

#### *SET NULL*

It is used in conjunction with ON DELETE or ON UPDATE. It means that the child data is set to NULL when the parent data is deleted or updated.

#### *SET DEFAULT*

It is used in conjunction with ON DELETE or ON UPDATE. It means that the child data is set to their default values when the parent data is deleted or updated.

**Example:-**

Create Table products

```
( product_id int primarykey,  
  product_name varchar(50)not null,  
  category varchar(25)  
);
```

Create Table inventory

```
( inventory_id int primary key,  
  product_id int not null,  
  quantity int,  
  min_level int,  
  max_level int,  
  Constraint fk_inv_product_id  
  Foreign key (product_id)  
  References products(product_id)  
  on delete cascade  
);
```

In this foreign key example, we've created our parent table as the products table. The products table has a primary key that consists of the product\_id field.

Next, we've created a second table called inventory that will be the child table in this foreign key with cascade delete example. We have used the CREATE TABLE statement to create a foreign key on the inventory table called fk\_inv\_product\_id. The foreign key establishes a relationship between the product\_id column in the inventory table and the product\_id column in the products table.

For this foreign key, we have specified the ON DELETE CASCADE clause which tells SQL Server to delete the corresponding records in the child table when the data in the parent table is deleted. So in this example, if a product\_id value is deleted from the products table, the corresponding records in the inventory table that use this product\_id will also be deleted.

Create a foreign key with cascade delete - Using Alter Table statement:-

Syntax:-

Alter table child\_table add constraint fk\_name

Foreign key (child\_col1, child\_col2,...)

References parent\_table (parent\_col1, parent\_col2,..)

On delete cascade;

E.g:-

Alter table inventory add constraint fk\_inv\_product\_id

Foreign key (product\_id) references products (product\_id)

On delete cascade;

In this foreign key example, we've created a foreign key on the inventory table called fk\_inv\_product\_id that references the products table based on the product\_id field.

For this foreign key, we have specified the ON DELETE CASCADE clause which tells SQL Server to delete the corresponding records in the child table when the data in the parent table is deleted. So in this example, if a product\_id value is deleted from the products table, the corresponding records in the inventory table that use this product\_id will also be deleted.

## Check Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

Example:-

The following SQL creates a new table called CUSTOMERS and adds four columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER below 18 years

Create table Customers

```
(  
id int not null,  
name varchar (20)not null,  
age int not nullcheck (age >= 18),  
salary decimal (5, 2),primary key (id)  
);
```

In the above example we are created Check constraint on asingle column at the time of table creation.

At Table Level:-

Create table Customers

```
(  
id int not null,  
name varchar (20)not null,  
age int not null,  
salary decimal (5, 2),  
primary key (id),  
Constraint chk_cust check (age >= 18 and and id>=0)  
);
```

In the above example we are created Check constraint on multiple columns at table level.

By Using Alter Statement:-

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following

Syntax:-

```
Alter table<Table_Name>add constraint<Constraint_Name>check  
<(Condition)>;
```

Eg:-

```
Alter table customers add constraint  
Ch_1 check (AGE >= 18 );
```

Drop Constraint:-

Syntax:

```
Alter table<Table_Name>drop constraint<Constraint_Name>;
```

Eg:-

```
Alter table customers drop constraint Ch_1;
```

## **Functions**

- Functions are nothing but set of statements which will perform some operations and returns a value to the user.
- The return type may be integer, string, float value.

Functions are classified into 2 types

- System defined functions (Inbuilt)
- User defined functions

### **Note:-**

Based on return result these function are again classified into 2 types

- Single Row
- Multi Row

Single Row:-

- The function which will process on single row at a time and return only one value is called as Single row function.

Multi Row:-

- The functions which will process on multiple rows at a time and return only single value is called multi row function.
- Aggregate functions belongs to multi-function  
( E.g:- Max, Min, Count, Sum & Avg..)

### **System defined functions:-**

SQL will support some built-in functions these are

- Mathematical / Numerical type
- Character / String
- Date and Time
- Aggregative / Group functions

Syntax:-

Select <Function\_Name> (Value /Expression);

**Mathematical Functions:-**

The functions which will take input as number and returns integer value. Means these are working on numerical values.

1. ABS(number) -- Absolute

It will return only positive values of given Number within the ()

E.g:-

Select abs(-3);

o/p : 3

2. SQRT(Number) -- Square root

This function is used to find the Square root value of given Number.

E.g:-

Select SQRT(100);

o/p : 10

3. POWER(base no / exponent no)

This is used to find the power of the given number.

E.g:-

Select POWER(3,6);

o/p: 729

4. ROUND(Number, Position)

This is used to round the given number.

E.g:-

Select round(996.66479,2);

o/p : 996.66000

5. CEILING(Number)

This function returns a value which is greater or equal to the given number.

E.g:- select CEILING(9.045); ----- o/p 10

Select CEILING(-4.21); ----- o/p -4

Select CEILING(9.0003); ----- o/p 10

Select CEILING(-4.002); -----o/p -4

6. FLOOR(Number)

This function return a value which is less than or equal to the given number.

E.g:- select FLOOR(8.0); ----o/p 8

Select FLOOR(8.001); --- o/p 8

Select FLOOR(-8.0); --- o/p -8

Select FLOOR(-8.0001); --- o/p -8

7. SIGN(Number)

This function will return `+1` for any `+ve` number, `0` for neutral value & `-1` for any `-ve` number.

E.g:- select SIGN(2);--- o/p '1'

Select SIGN(-2);--- o/p '-1'

Select SIGN(0);-----o/p '0'

8. PI()

This function is used to get the `PI` value of Constant.

E.g:-select PI(); --- o/p 3.14159265358979

9. SIN(Number)

This function is used to get the sin value given in degree's.

E.g:- select sin(90);--- o/p 0.893996663600558

Select SIN(0);----- o/p 0

Select sin(180); ---o/p -0.80115263573383

10. TAN(Number)

This function is used to get the Tan value given in degree's

E.g:- select tan(90);-- o/p -1.99520041220824

Select tan(0); --- o/p 0

11. ATAN(Number)

This function returns the given number arctangent value.

E.g:- SELECT ATAN(90);--- o/p 1.55968567289729

### **String Functions:-**

These functions will take input as string and returns string / Integer value.

1. Len(String):-

It will return the length of the given string. This will take the input as string and return integer value.

E.g:-

Select len('Machavaram'); -- -- o/p 10

2. Right(String,num):-

This will return the specified number of characters from right side of the string.

E.g:-

Select RIGHT('Machavaram',5); ----- o/p varam



**3. LEFT(String,num):-**

This will return the specified number of characters from left side of the string.

E.g:-

```
Select LEFT('Machavaram',5);--- o/p Macha
```

**4. UPPER(String):-**

This will convert given string to Upper case from Lower case.

E.g:-

```
Select UPPER('saimachavaram'); ---o/p SAIMACHAVARAM
```

**5. ASCII(Char):-**

This will return the ASCII values of given Character. The value will change based on case sensitive.

E.g:-

```
Select ASCII('z'); -- o/p 122
Select ASCII('m'); -- o/p 109
Select ASCII('M'); -- o/p 77
```

**6. CHAR(Num):-**

This will return the Character of given ASCII value.

E.g:-

```
Select CHAR(83);-- o/p S
Select CHAR(101);--- o/p e
```

**7. LTRIM():-**

This function will Trim's the left side space of the given string in Expression.

E.g:-

```
Select LTRIM(' Sai@Machavaram');
--- o/p Sai@Machavaram
```

**8. RTRIM():-**

This function will Trims the Right side spaces of the given string in Expression.

E.g:-

```
Select RTRIM('Sai@Machavaram ');
-- o/p Sai@Machavaram
```

**9. CHARINDEX():-**

This function will return the index value of the specified character from the given string in expression.

E.g:-

Select CHARINDEX('D','INDIA'); --- o/p 3

**10. REPLACE():-**

By using this function we can replace the existing Characters with new Characters in the given expression string.

E.g:-

Select REPLACE('JACK','J','BL'); -- o/p BLACK

**11. REVERSE():-**

This function will reverse the characters in the given string expression.

E.g:-

Select REVERSE('SAI');-- o/p IAS

**12. REPLICATE():-**

This function is used to repeats the characters in the given expression as per the specified number of times.

E.g:-

Select REPLICATE('SAI',6); -- o/p SAISAISAISAISAI

**13. SUBSTRING():-**

This function contains 3 arguments those are Expression, Starting position character and length of the characters. This function is used to returns the required sub string from the given string expression.

E.g:-

->select substring('Machavaram',4,3); -- o/p hav

-> select SUBSTRING('Machavaram@Email.com',1,2)+  
replicate('\*',6)+ SUBSTRING('Machavaram@Email.com',  
CHARINDEX('@','Machavaram@Email.com')+1,len('machavaram@emai  
l.com')-CHARINDEX('@','machavaram@gmail.com'));  
O/p Ma\*\*\*\*\*Email.com

**14. CONCAT():-**

This function will merge (add) the two or more string expressions.

E.g:-

Select concat ('SAI','MACHAVARAM');

-- o/p SAI.MACHAVARAM

(This will work in SQL 2012 are higher versions of SQL)

**15. SPACES():-**

This function will provide the spaces between two or more in the string expression.

E.g:-

```
select ('Sai'+SPACE(30)+'Machavaram');
```

o/p Sai

Machavaram

**16. STUFF():-**

This function is similar to replace function, Inserts a replacement expression, at the specified start position, along with removing the characters specified in the using length parameter.

E.g:-

```
Select STUFF('Machavaram@Email.com',2,4,'#####');
```

o/p:M#####varam@Email.com

**Date & Time Functions:-**

These functions are used to perform operations on Current System Date & Time. Which works on given expression.

**1. GETDATE()**

This function returns the current Date and Time of the system.

E.g:-

```
Select GETDATE(); -- o/p 2017-02-03 10:54:00.777
```

**2. MONTH()**

This will return the current system month.

E.g:-

```
Select MONTH(GETDATE());-- o/p 2
```

**3. DAY()**

This will return the current system day.

E.g:-

```
Select DAY(GETDATE());-- o/p 3
```

**4. YEAR()**

This will return the current system Year.

E.g:-

```
Select YEAR(GETDATE());-- o/p 2017
```

5. To get current system Date only(Day-Month-Year)

This will return the current system Date.

E.g:-

```
Select DAY(GETDATE()),MONTH(GETDATE()),YEAR(GETDATE());  
o/p 3 2 2017 @@
```

6. DATENAME()

This function will return the name of Day, Month intervals in the given expression.

E.g:-

```
Select DATENAME(DW,GETDATE()); o/p Friday  
Select DATENAME(MM,GETDATE()); o/p February  
Select DATENAME(DAYOFYEAR,GETDATE()); o/p 34  
Select DATENAME(QUARTER,GETDATE()); o/p 1  
Select DATENAME(HOUR,GETDATE()); o/p 12
```

7. DATEADD()

This function will return the values of how many days, months & years you are adding to the given expression.

E.g:-

```
Select DATEADD(dd,15,GETDATE());o/p 2017-02-18 12:16:03.470  
Select DATEADD(MM,7,GETDATE()); o/p 2017-09-03 12:19:58.640  
Select DATEADD(YY,02,GETDATE());o/p 2019-02-03 12:19:58.640
```

8. DATEPART()

This function returns date, month and year path values from the given date expression.

E.g:-

```
select DATEPART (DD,GETDATE()); o/p 3  
select DATEPART (mm,GETDATE()); o/p 2  
select DATEPART (YYYY,GETDATE());o/p 2017
```

9. DATEDIFF()

This function returns the differences between starting and ending date expressions.

E.g:- 

```
Select datediff(DD,'2016-9-18',GETDATE()); o/p 138  
Select datediff(mm,'2016-9-18',GETDATE()); o/p 5  
Select datediff(YYYY,'2016-9-18',GETDATE()); o/p 1
```

10. ISDATE ()

This function determines whether an input expression is a valid date. If it is valid returns \_1\_ invalid returns \_0\_.

E.g:-

```
Select ISDATE('15/04/2017'); o/p 0  
Select ISDATE('3/02/2017'); o/p 1
```

**Aggregate / Group Functions:-**

Aggregate functions perform a calculation on a set of values and return a single value.

--> Create a Emp table like bellow to perform these functions.

Eid	Name	Age	Gender	Dept	Address	Salary	Cell_No	EMail
1	Aashrith	28	M	IT	Ongole	32000.00	9966640779	parasara189@gmail.com
2	Mahathi	32	F	HR	Hyderabad	47000.00	6878456520	mahathi_2@gmail.com
3	Karthik	26	M	DBA	Mumbai	31500.00	8456876520	karthik_3@ymail.com
4	Saamya	25	F	HR	Chennai	33000.00	5656878420	saamya_4@ymail.com
5	Saanya	25	F	HR	Chennai	33000.00	2068784520	ssanya_5@ms.net
6	Agasthya	29	M	IT	Lucknow	34500.00	5656878420	agasthya_6@ms.net

**1. SUM():-**

This function returns the sum of given group of values.

E.g:-

Select sum(Salary) from Tbl\_Emp;

O/p : 211000.00

**2. AVG():-**

AVG returns the average of the values in expression. The expression must contain numeric values. Null values are ignored.

E.g:-

Select avg (Salary) from Tbl\_Emp;

O/p: 35166.6666

**3. MAX():-**

This function will return the Maximum value of given group of values.

E.g:-

Select max (salary) from tbl\_emp;

O/p: 47000.00

**4. MIN():-**

This function will return the Minimum value of given group of values.

E.g:-

Select min (salary) from Tbl\_Emp;

O/p: 31500.00

## 5. COUNT(\*):-

This function counts all values including duplicate, Null values in the table.

E.g:-

Select count (\*) from Tbl\_Emp;

O/p: 6

## 6. COUNT(Distinct&lt;Column\_Name&gt;):-

This function returns unique count of the mentioned column.

E.g:-

Select count (distinct Name) from tbl\_emp;

O/p : 6

## 7. COUNT(&lt;Column\_Name&gt;):-

This function counts all the count including Duplicate & Null values on the mentioned column.

E.g:-

Select count (name) from tbl\_emp;

O/p : 6

## 8. STDEV():-

Returns the standard deviation of all values in expression. Stdev ignores any NULL values.

E.g:-

Select stdev (salary) from Tbl\_Emp;

O/p: 5887.84057755189

## **Joins**

Whenever we want to fetch data from one or more database tables based on common field (column) we use Joins. Which is joined to appear as single set of data. **Join** Keyword is used in SQL queries for joining two or more tables. A table can also join to itself known as, Self Join.

### **Why SQL JOIN is used**

- If you want to combine two or more table then SQL JOIN statement is used. It combines rows of that tables in one table and one can retrieve the information by a SELECT statement.
- The joining of two or more tables is based on common field between them.

### **Types of Joins**

There several Types of Joins such as

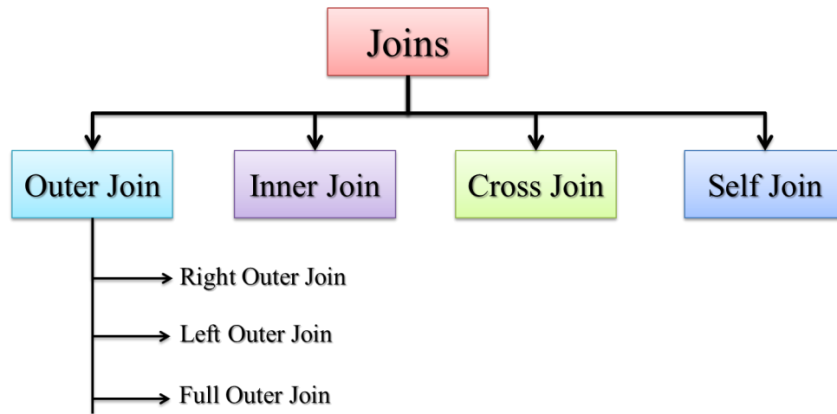
- Inner Join
- Outer Join
- Cross Join
- Self-Join
- Natural Join

Outer Join can further be categorize into 3 more categories

- Right Outer Join / Right join
- Left Outer Join / Left join
- Full Outer Join. / Full join

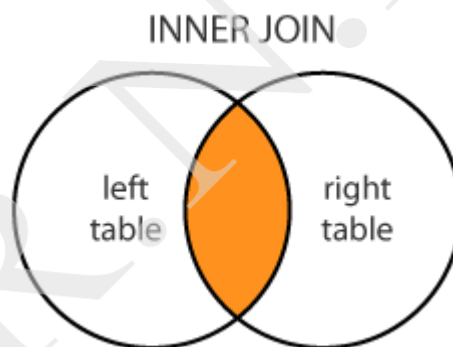
Apart from types of join we also have two join conditions these are

- *Equi Joins*
- *Non-Equi Joins*



### Inner join:-

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.



Syntax:-

The basic syntax of INNER JOIN

- 1) SELECT table1.column1, table2.column2...  
FROM table1  
JOIN table2  
ON table1.common\_field = table2.common\_field;  
(OR)
- 2) SELECT table1.column1, table2.column2...  
FROM table1  
INNER JOIN table2  
ON table1.common\_field = table2.common\_field;



**Example:-**

Let's assume we have 2 tables i.e Emp\_Tbl & Dept\_Tbl like bellow

**Emp\_Tbl (Child Table)****Dept\_Tbl (Parent Table)**

Eid	Name	Salry	Dept_ID
1	Aashrith	19000.00	1
2	Mahathi	20000.00	2
3	Karthik	25000.00	3
4	Saamya	18000.00	2
5	Scott	19000.00	3
6	James	18000.00	1
7	Wasif	19000.00	NULL
8	Ramesh	18000.00	NULL

id	Dept_Nmae	Dept_Location	Dept_Head
1	IT	Hyderabad	Rama
2	HR	Chennai	sita
3	Finance	Mumbai	Patel
4	Sales	Amaravati	Yadav

Dept\_Tbl table creation syntax:-

Create table Dept\_Tbl

(

Dept\_id int constraint Dept\_ID\_pk primary key,

Dept\_Name varchar(20),

Dept\_Location varchar(20),

Dept\_Head varchar(20)

)

Emp\_Tbl table creation syntax:-

Create table Emp\_tbl

(

Eid int constraint emp\_eid\_pk PRIMARY KEY,

Name varchar(20),

Salary money,

Dept\_ID int constraint Emp\_id\_fk references Dept\_tbl(Dept\_id)

)

Inserting Values Syntax:-

Insert into<Table\_Name>values(Column 1 value, column 2 value...);

Inner join result would be like

Eid	Name	salry	Dept_Nmae
1	Aashrith	19000.00	IT
2	Mahathi	20000.00	HR
3	Karthik	25000.00	Finance
4	Saamya	18000.00	HR
5	Scott	19000.00	Finance
6	James	18000.00	IT

Query for above Inner Join:

```
select Eid, Name, Salary, Dept_Name from Emp_tbl  
inner join Dept_Tbl on Emp_tbl.Dept_ID=Dept_Tbl.id;
```

- In the first line of our syntax we have our **SELECT** statement where you specify all those columns from both the participating table (Emp\_Tbl and Dept\_Tbl) whose data you want to fetch in your result set. The SELECT statement is followed by **FROM** keyword.
- In the second line of our syntax we have our **JOIN clause** which is INNER join as obvious. You can either write INNER JOIN or simple JOIN as both are permissible and perform the same task. On both side of our Join clause we have our tables which are Emp\_Tbl and Dept\_Tbl.

INNER JOIN with ON clause *Query:-*

```
Select Name, Dept_Name from Emp_tbl inner join Dept_Tbl  
ON(Emp_tbl.Dept_id = Dept_Tbl.ID);
```

- Here in this query we are selecting emp name from Emp\_Tbl and dept name from Dept\_Tbl while in JOIN condition which in this case is the ON clause where we are comparing 'id' column of both the tables.
- As here in this query we are using ON join condition thus we can use any column as expression of ON clause as long as columns share same data types. The name of the column doesn't matter here.

### INNER JOIN with WHERE clause

- We use WHERE clause to limit the result of a Query, similarly you can use WHERE clause here with Inner join to do the same. Say you want to see the name and departments of only those employees who have a salary of less than 20000 For that you just have to add the where clause right after the JOIN condition in the query.

Query:-

- 1) Select Eid,Name,Dept\_Name from Emp\_tbl INNER JOIN Dept\_Tbl on Emp\_tbl.Dept\_id =Dept\_Tbl.id where Emp\_tbl.Salary < 20000;
- 2) Select Eid,Name,Dept\_Name from Emp\_tbl inner joinDept\_Tbl on Emp\_tbl.Dept\_id =Dept\_Tbl.id where Emp\_tbl.Salary < 20000 and Dept\_Tbl.id=1;

### INNER JOIN with ORDER BY clause.

- You can use ORDER BY clause if you want to sort the result returned by your query in Ascending or Descending order.
- ORDER BY clause by default sorts the result in ascending order. But if you want to arrange the result in Descending order then you have to specify it by using DESCENDING or DESC keyword with the ORDER BY clause.

Note: - Here ORDER BY clause must be the last statement of a query.

#### Example

Sort the result in ascending order according to Name column

Query

```
Select Eid,Name,Dept_Name from Emp_tbl inner joinDept_Tbl on  
Emp_tbl.Dept_id =Dept_Tbl.id where  
Emp_tbl.Salary > 20000 order by Name desc;
```

Sort the result in descending order according to emp\_name column.

Query

```
Select Eid,Name,Dept_Name from Emp_tbl inner joinDept_Tbl on  
Emp_tbl.Dept_id =Dept_Tbl.id where  
Emp_tbl.Salary > 20000 order by Name desc;
```

**Table Alias:**

- An Alias is shorthand for a table or column name.
- Aliases reduce the amount of typing required to enter a query.
- Complex queries with aliases are generally easier to read.
- Aliases are useful with JOINS and aggregates: SUM, COUNT, etc.
- An Alias only exists for the duration of the query.

**Example Queries:-**

```
Select Eid,Name,Dept_Name FROM Emp_tbl e inner join Dept_Tbl d
on e.Eid=d.id where e.Salary > 20000 and d.id=1;
```

- ➔ In the above query `_e'` is the alias name for `Emp_Tbl` , `_d'` is the alias name for `Dept_Tbl`

// WAQ to display Employee details whose salary range between 20000 and 25000 who are working in IT department.

**Query:-**

```
Select Eid,Name,Salary,Dept_Name from Emp_Tbl innerjoin Dept_Tbl
on Emp_tbl.Dept_id =Dept_tbl.id where dept_name='IT'and salary
between 20000 and 25000;
```

**Inner Join Examples with 3 Tables:-**

Let's create 3 tables like bellow

**Table1:-** create table student

(sno int primary key, sname varchar(20),addres svarchar(50));

**Tbale2:-**creat etable course

(cid int primary key, cname varchar(20));

**Table3:-**create table enrollment

(sno int foreign key references student(sno),cid int foreign key references course(cid));

Assume the data in the 3 Tables like, alias names like s,c,e.

Student s			Course c		Enrollment e	
sno	sname	address	cid	cname	sno	cid
101	Machavaram	Amaravathi	1	SQL	101	1
102	Krishna	Hyderbad	2	Oracle	102	3
103	Koti	Hyderbad	3	MySQL	103	4
104	Sai	Ongole	4	Dot Net	104	2

//WAQ to display student details along with course name.

```
select s.*,c.cname from student s inner join enrollment e on s.sno=e.sno
inner join course c on e.cid=c.cid;
```

//WAQ to display student details who enrolled for SQL

```
select s.sname from course c inner join enrollment e on c.cid=e.cid inner
join student s on e.sno=s.sno where c.cname='SQL';
```

//WAQ to display the student names who are enrolled for SQL & Dot Net

```
Select distinct(s.sname)from course c inner join enrollment e on
c.cid=e.cid inner join student s on e.sno=s.sno;
```

//WAQ to display the student names along with coursename who are staying in 'Ongole';

```
select s.sname from student s inner join enrollment e on s.sno=e.sno inner
join course c on e.cid=c.cid where s.address='Ongole';
```

//WAQ to display the student names who are enrolled for 'SQL' whose name starts with 'M';

```
select s.sname from course c inner join enrollment e on c.cid=e.cid inner
join student s on e.sno=s.sno where s.sname like'M%';
```

## Outer join

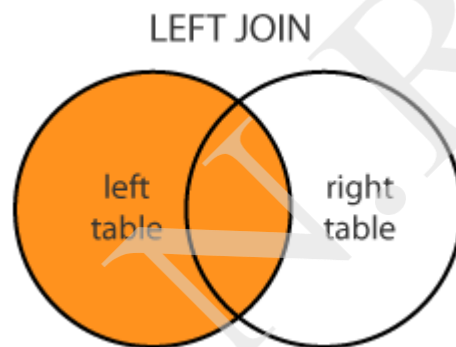
Outer Join can further be categorize into 3 more categories

- Left Outer Join / Left join
- Right Outer Join / Right join
- Full Outer Join. / Full join

Left join:-

The left outer join returns a result table with the matched data of two tables and also gives remaining rows of the left table and null for the right table's column.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.



Syntax:-

```
Select <column-names>
FROM <table-name1> Left Join <table-name2>
on <column-name1> = <column-name2>
where <condition>;
```

\* \_Outer' key word is an optional.

Ex: - Let's consider we have two tables like below

Emp\_Tbl (Child Table)

EId	Nmae	Salry	Dept_ID
1	Aashrith	19000.00	1
2	Mahathi	20000.00	2
3	Karthik	25000.00	3
4	Saamya	18000.00	2
5	Scott	19000.00	3
6	James	18000.00	1
7	Wasif	19000.00	NULL
8	Ramesh	18000.00	NULL

Dept\_Tbl (Parent Table)

id	Dept_Nmae	Dept_Location	Dept_Head
1	IT	Hyderabad	Rama
2	HR	Chennai	sita
3	Finance	Mumbai	Patel
4	Sales	Amaravati	Yadav

**Example Queries:-**

1) select Eid, Name, Salary, Dept\_Name from Emp\_tbl  
left join Dept\_Tbl on Emp\_tbl.Dept\_ID= Dept\_Tbl.id;

It will return the matching records (whose Dept\_id & Id columns is Equal) of Eid, Name, Salary & Dept\_Name from the both tables and also it will return non matching data from Left Table.

2) Select Eid, Name, Dept\_Name FROM Emp\_tbl left join Dept\_Tbl  
on Emp\_tbl. Dept\_id =Dept\_Tbl.id where  
Emp\_tbl.Salary > 20000;

It will return the matching records (whose Dept\_id & Id columns is Equal and whose salary is greater than 20000) of Eid, Name, Salary & Dept\_Name from the both tables and also it will return non matching data from Left Table.

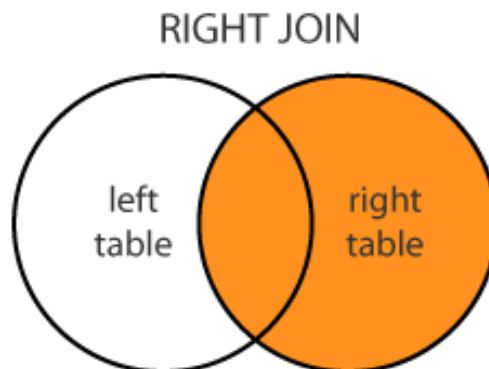
3) SELECT Eid, Name, Dept\_Name FROM Emp\_tbl left  
join Dept\_Tbl on Emp\_tbl. Dept\_id =Dept\_Tbl.id where  
Emp\_tbl.Salary > 20000 order by Name desc;

**Right join:-**

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Right Join and Right Outer Join are the same.



Syntax:-

```
SELECT <column-names>
      FROM <table-name1>RIGHT JOIN <table-name2>
      ON <column-name1> = <column-name2>
WHERE <condition>;
```

\* '\_Outer' key word is an optional.

Ex: - Let's consider we have two tables like below

Emp\_Tbl (Child Table)

Eid	Nmae	Salry	Dept_ID
1	Aashrith	19000.00	1
2	Mahathi	20000.00	2
3	Karthik	25000.00	3
4	Saamya	18000.00	2
5	Scott	19000.00	3
6	James	18000.00	1
7	Wasif	19000.00	NULL
8	Ramesh	18000.00	NULL

Dept\_Tbl (Parent Table)

id	Dept_Nmae	Dept_Location	Dept_Head
1	IT	Hyderabad	Rama
2	HR	Chennai	sita
3	Finance	Mumbai	Patel
4	Sales	Amaravati	Yadav

Example Queries:-

- 1) select Eid, Name, Salary,Dept\_Name from Emp\_tbl  
Right join Dept\_Tbl on Emp\_tbl.Dept\_ID= Dept\_Tbl.id;

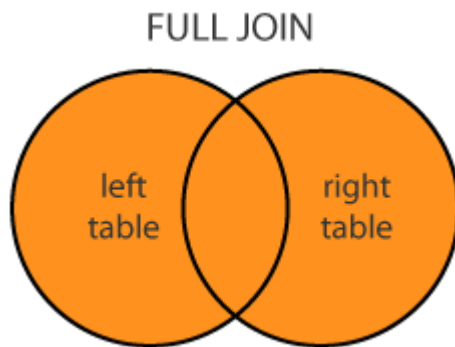
It will return the matching records (whose Dept\_id & Id columns is Equal) of Eid, Name, Salary & Dept\_Name from the both tables and also it will return non matching data from Right Table.

- 2) select Eid,Name,Dept\_Name from Emp\_tbl Right join Dept\_Tbl on  
Emp\_tbl.Eid=Dept\_Tbl.id where Emp\_tbl.Salary > 20000;
- 3) select Eid,Name,Dept\_Name from Emp\_tbl Right join Dept\_Tbl on  
Emp\_tbl.Eid=Dept\_Tbl.id where Emp\_tbl.Salary > 20000 order by  
Name desc;



Full join:-

- FULL JOIN returns all matching records from both tables whether the other table matches or not.
- FULL JOIN can potentially return very large datasets.
- FULL JOIN and FULL OUTER JOIN are the same.



Syntax:-

```
SELECT <column-names>  
      FROM <table-name1> FULL JOIN <table-name2>  
      ON <column-name1> = <column-name2>  
WHERE <condition>;
```

\* '\_Outer' key word is an optional

Example Queries:-

1) select Eid, Name, Salary, Dept\_Name from Emp\_tbl  
Full join Dept\_Tbl on Emp\_tbl.Dept\_ID= Dept\_Tbl.id;

It will return the matching records both tables as well as non-matching records from the Left & right table too.

2) Select Eid, Name, Dept\_Name from Emp\_tbl Full join Dept\_Tbl on  
Emp\_tbl.Eid=Dept\_Tbl.id where Emp\_tbl.Salary > 20000;

3) Select Eid, Name, Dept\_Name from Emp\_tbl Full join Dept\_Tbl on  
Emp\_tbl.Eid=Dept\_Tbl.id where Emp\_tbl.Salary > 20000 order by  
Name desc;

**Cross Join:-**

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

E.g.:- There are 2 'M' rows in Left Table , 'N' rows in Right Table then Cartesian product = 'M \* N' rows.

**Syntax:-**

- 1) Select <column names>from <table1>cross join <table 2>;
- 2) Select <column names>from <table1>cross join <table 2>  
WHERE (expression) order by <column names>;
- 3) Select <column names>from <table1>, <table 2> WHERE  
(expression) order By <column names>;

**Example Queries:-**

- 1) select Eid, Name, Salary,Dept\_Name from Emp\_tbl  
cross join Dept\_Tbl;
- 2) select Eid,Name,Dept\_Name FROM Emp\_tbl cross join Dept\_Tbl  
where Emp\_tbl.Salary > 20000;
- 3) select Eid,Name,Dept\_Name from Emp\_tbl cross join Dept\_Tbl  
where Emp\_tbl.Salary > 20000 order by Name desc;

## Self-Join:-

The SQL SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement. This is also useful for comparisons within a table.

It can be classified any type of Join (Inner, Outer, Cross).

### Syntax:-

```
SELECT <column-names> FROM <Table-name >T1 JOIN  
<Table-name> T2 WHERE (condition);
```

Example:-

Let's assume we have a table i.e Self\_Join\_tbl like bellow.

```
Create table Self_Join_tbl  
(  
  Empid int,  
  Name Varchar(20),  
  Mgr_id int,  
);
```

--Inserting multiple values at a time:-

```
Insert into Self_Join_tbl  
(Empid,Name,Mgr_id)values  
(1,'Viswanath',3),  
(2,'Sekhar',2),  
(3,'Kumar',Null),  
(4,'Raghu',1);
```

--The table is:

Empid	Name	Mgr_id
1	Viswanath	3
2	Sekhar	2
3	Kumar	NULL

## Example Queries:-

1) select E.Name as Employee , M.Name as Manager from Self\_Join\_tbl E left join Self\_Join\_tbl M on e.Mgr\_id=m.Empid;

In the above example treating the same table as 2 different table's i.e Self\_Join\_tbl E as 'Employee' & Self\_Join\_tbl M as 'Manager' tables. In the 'Employee' table I want to take the Mgr\_id Column and look that up in the 'Manager' table in the EmpId column. The actual table joining itself and it will return's all the matching records and also will return non matching records from Employee i.e Kumar .

Employee	Manager
Viswanath	Kumar
Sekhar	Sekhar
Kumar	NULL
Raghu	Viswanath

2) select E.Name as Employee , M.Name as Manager from Self\_Join\_tbl E inner join Self\_Join\_tbl M on e.Mgr\_id=m.Empid;

In the above example it will return only matching record's, it won't return Null value record i.e Kumar.

Employee	Manager
Viswanath	Kumar
Sekhar	Sekhar
Raghu	Viswanath

3) select E.Name as Employee , M.Name as Manager from Self\_Join\_tbl E inner join Self\_Join\_tbl M on e.Mgr\_id=m.Empid where e.Empid=2;

It will return only 'Sekhar' record only why because we are used where condition.

<u>Employee</u>	<u>Manager</u>
Sekhar	Sekhar

**Equi-Joins:-**

- It is used to displaying the match records from both the tables by using equal (=) operator.
- While working with equi join a common column must exists between the both tables.

Example:-

```
select e.*,d.*from Emp_tbl e ,Dept_Tbl d where e.Eid=d.id;
```

**Non-Equi-Joins:-**

- It is used to display the data from both the tables without using Equal (=) operator.
- The SQL Non Equi Join uses comparison operator instead of the equal sign like >, <, >=, <=along with conditions.

Example:-

- 1) select e.\*,d.\*from Emp\_tbl e , Dept\_Tbl d where e.Salary>=20000;
- 2) select Eid,Name,Dept\_Name,Dept\_Head from Emp\_tbl e, Dept\_Tbl d where e.Dept\_ID<>d.id;

It will return Eid,Name,Dept\_Name& Dept\_Head from the both table's whoes Dept\_id in Emp\_Tbl is not equal to id in Dept\_Tbl table column.

**Natural Join:-**

It is similar to Equi join but in natural join we will avoid the duplicate column name from the result set.

E.g:-

```
Select e.eid,e.name,d.id, d.dept_location from emp_tble, dept_tbl d where e.Eid=d.id
```

**Advanced Joins in SQL Server****Non Matching row's from the Left Table:-**

If you want to get only Non-matching rows from the Left Table.

**Query:-**

```
select eid,Name,salary from Emp_tbl E left join Dept_Tbl D on  
e.Dept_ID=d.id where D.id is null;
```

**Non Matching row's from the Right Table:-**

If you want to get only Non-matching rows from the Right Table.

**Query:-**

```
select Name,Salary,Dept_Name from Emp_tbl E Right join  
Dept_Tbl D on e.Dept_ID=d.id where E.Dept_ID is null;
```

**Non Matching row's from the Left & Right Table:-**

If you want to get only Non-matching rows from the Left&Right Table.

**Query:-**

```
select Name,Salary,Dept_Name from Emp_tbl E Full join Dept_Tbl  
D on e.Dept_ID=d.id where E.Dept_ID is null or d.id is null;
```

**Note:- In SQL Server you never ever use „=" operator when comparing NULL values. Always"s use only „IS" keyword.**

## **Normalization of Database**

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable (unwanted) characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

### ***Problem without Normalization***

- Disk space wastage.
- It becomes difficult to handle and update the database, without facing data loss.
- Insertion, Updation and Deletion Anomalies are very frequent if Database is not normalized, at that time these DML queries execution can become slow.
- Data inconsistency (irregularity).

To understand these anomalies let us take an example of Employee table.

### **Employee table (without Normalization)**

<b>EID</b>	<b>Name</b>	<b>Gender</b>	<b>Salary</b>	<b>Dept Name</b>	<b>Dept Head</b>	<b>Dept Location</b>
1	Aashrith	Male	9800	IT	Rama	Hyderabad
2	Saamya	Female	8700	HR	Seeta	Chennai
3	Basha	Male	6750	IT	Rama	Hyderabad
4	John	Male	7189	IT	Rama	Hyderabad
5	Radha	Female	9189	HR	Seeta	Chennai

Normalized Employee table design

E ID	Name	Gender	Salary	Dept ID
1	Aashrith	Male	9800	1
2	Saamyra	Female	8700	2
3	Basha	Male	6750	1
4	John	Male	7189	1
5	Radha	Female	9189	2

Dept ID	Dept Name	Dept Head	DEpt Location
1	IT	Rama	Hyderabad
2	HR	Seeta	Chennai

- There are Six Normal Forms (NF) are there in Database, they are 1NF to 6NF. Most Database in the real world we will use 3NF.
- For each NF certain rules are there.



**First Normal Form (1NF):-**

A table is said to be in 1NF, if

- The data in each column should be Atomic. No multiple values, separated by comma.
- The table does not contain any repeating column groups.
- Identify each record uniquely using primary key.

Non Atomic Employee column

Dept Name	Employee
IT	Aashrith, Basha, Johan
HR	Saamy, Seeta

It is not possible to select, Insert, Update, Delete just one employee }

No repeating Column groups:-

Dept Name	Employee1	Employee2	Employee3
IT	Aashrith	Basha	Johan
HR	Saamy	Seeta	

1. In IT department if more than 3 employees Table structure need to change.
2. If less than 3 employees wasted of Disk space.

To achieve 1NF we are splitting the above table like bellow

Primary Key

Dept ID	Dept Name
1	IT
2	HR

Foreign Key

Dept ID	Employee
1	Aashrith
1	Basha
1	John
2	Saamy
2	Seeta



**Second Normal Form (2NF)**

- The table meets all the conditions of 1NF and there must be no partial dependences of any of the columns on the primary key.
- Move redundant data as separate table.
- Create relationship between two tables using Foreign keys.

Employee table (without Normalization)

EID	Name	Gender	Salary	Dept Name	Dept Head	Dept Location
1	Aashrith	Male	9800	IT	Rama	Hyderabad
2	Saamya	Female	8700	HR	Seeta	Chennai
3	Basha	Male	6750	IT	Rama	Hyderabad
4	John	Male	7189	IT	Rama	Hyderabad
5	Radha	Female	9189	HR	Seeta	Chennai

Employee table design in 2NF

Dept ID	Dept Name	Dept Head	DEpt Location
1	IT	Rama	Hyderabad
2	HR	Seeta	Chennai

E ID	Name	Gender	Salary	Dept ID
1	Aashrith	Male	9800	1
2	Saamya	Female	8700	2
3	Basha	Male	6750	1
4	John	Male	7189	1
5	Radha	Female	9189	2

We are established relationship between to tables by using Forign key.

**Third Normal Form (3NF)**

A Table is said to be in 3NF, if the table

- Meets all the conditions of 2NF.
- Does not contain columns that are not fully dependent upon the primary key.

EID	Name	Gender	Salary	Annual Salary	Dept Name	Dept Head	Dept Location
1	Aashrith	Male	9800	<del>117600</del>	IT	Rama	Hyderabad
2	Saamya	Female	8700	<del>104400</del>	HR	Seeta	Chennai
3	Basha	Male	6750	<del>81000</del>	IT	Rama	Hyderabad
4	John	Male	7189	<del>86268</del>	IT	Rama	Hyderabad
5	Radha	Female	9189	110268	HR	Seeta	Chennai

If you observe the above table, in the table all the columns except Annual salary remain all are fully dependent on primary key i.e EID. The Annual Salary column depends on Salary it's a computed column, there is no need to store these types of columns in table you can remove from the table or else keep in a separate table.

Moving to a separate table Example:-

EID	Name	Gender	Salary	Dept Name	Dept Head	Dept Location
1	Aashrith	Male	9800	IT	Rama	Hyderabad
2	Saamya	Female	8700	HR	Seeta	Chennai
3	Basha	Male	6750	IT	Rama	Hyderabad
4	John	Male	7189	IT	Rama	Hyderabad
5	Radha	Female	9189	HR	Seeta	Chennai

EID	Name	Gender	Salary	Dept ID
1	Aashrith	Male	9800	1
2	Saamya	Female	8700	2
3	Basha	Male	6750	1
4	John	Male	7189	1
5	Radha	Female	9189	2

Dept ID	Dept Name	Dept Head	DEpt Location
1	IT	Rama	Hyderabad
2	HR	Seeta	Chennai

If observe the above table that's not in 3NF, in all the columns Dept name, Dept Head, Dept Location are not fully dependent on primary key. Dept Head, Dept Location are dependent on Dept Name. we are breaking the above table into 2 tables

The advantage of removing transitive dependency is,  
Amount of data duplication is reduced.  
Data integrity achieved.

## **Indexes**

Generally every database server will perform 2 types of searching mechanism for retrieving the data from the Tables

- Table Scan
- Index Scan

### **Table Scan:-**

- If there is no index to help the query then the query engine, checks every row in the table from the beginning to end. This is called as Table scan. Table scan is bad for performance.
- It is a difficult mechanism of every database. In this mechanism the database server is searching on entire structure of the table for required data.
- So that it will take time consuming and application performance will slow.
- So to overcome the above drawbacks we used Index Scan Mechanism.

### **Index Scan:-**

- Indexes are special lookup tables that the database search engine can use to speed up data retrieval without reading the whole table. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.
- For example, if you want to reference all 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. If you don't have an Index, to locate a specific chapter in the book, you will have to look at every page of the book.
- In a similar way Table & View Indexes can help the query to find the data quickly.
- 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.

**Types of Indexes:-**

1. Clustered Index
2. Non-Clustered Index

**Syntax to Create an Index**

```
Create index <Index_Name> on  
    <table_Name>  
    (Column_Name [ASC | DESC]);
```

**Clustered Index:-**

- A clustered index determines the physical order of data in a table. For this reason a table can have only one clustered index.
- However the index can contain multiple columns(a composite index)
- Whenever we apply Primary key constraint on a column in a table, then automatically clustered index will applied on primary key column and clustered Index will arrange the data in Ascending order.
- We can apply only one clustered index on a single table.

Example:-

```
Create table [Tbl_Employee]  
(  
    [Id] int primary key,  
    [Name] varchar(20),  
    [Salary] money,  
    [Gender] char,  
    [City] varchar(20)  
)
```

By the above query we created a table with name `_Tbl_Employee` and we applied Primary key constraint also for `_Id` column , when ever we applied primary key it automatically creates Clustered Index on Primary key column.

To verify the Index:-

```
Execute sp_helpindex Tbl_Employee;
```

With the above query it will display the all Index's information which are assigned on that particular Table.

Creating Clustered Index Explicitly:-

Syntax:-

Create clustered index <Index\_Name> on

<Table\_Name>

(Colum\_Name ASC | Desc, Column-name ASC | Desc);

Eg:-

Create clustered index IX\_Tbl\_Employe\_Gender\_Salary on  
Tbl\_Employee (Gender Desc, Salary ASC);

Clustered Indexed Table would be like below after adding index's :-

<b>Id</b>	<b>Name</b>	<b>Salary</b>	<b>Gender</b>	<b>City</b>
<b>5</b>	Karthik	21250	M	Hyderabad
<b>6</b>	Agasthya	21250	M	Lucknow
<b>1</b>	Aasrith	32540	M	Ongole
<b>3</b>	Samhitha	19200	F	Chennai
<b>4</b>	Akshara	21250	F	Banglore
<b>2</b>	Mahathi	32540	F	Hyderabad
<b>7</b>	Saanya	32540	F	Chennai

### **Non- Clustered Index:-**

- A Non clustered index is analogous to an index in a textbook. The data is stored in one place , the index in another place. The index will have pointers to the storage location of data.
- Whenever we are applying Unique key constraint on a column in a table then it automatically creates Non clustered index will be applied on Unique column.
- We can apply more than one non clustered index on a single table.
- Non clustered index will not arrange the data in Ascending order.

Syntax:-

Create Non Clustered Index <Index\_Name> on <Table\_Name>  
(Column-Name);

Create Non Clustered Index IX\_Tbl\_Employee\_Name on  
Tbl\_Employee(Name);



<b>ID</b>	<b>Name</b>	<b>Salary</b>	<b>Gender</b>	<b>City</b>
5	Karthik	21250	M	Hyderabad
6	Agasthya	21250	M	Lucknow
1	Aasrith	32540	M	Ongole
3	Samhitha	19200	F	Chennai
4	Akshara	21250	F	Bangalore
2	Mahathi	32540	F	Hyderabad
7	Saanya	32540	F	Chennai

<b>Name</b>	<b>Row Address</b>
Karthik	Row Address
Agasthya	Row Address
Aasrith	Row Address
Samhitha	Row Address
Akshara	Row Address
Mahathi	Row Address
Saanya	Row Address

**NOTE:-**

- Only one clustered index per table, where as you can have more than one Non clustered index.
- Clustered index is faster than a non-clustered index , because the clustered index has to refer back to the table , if the selected column is not present in the index.
- Clustered index determines the storage order of rows in the table, and hence doesn't require additional disk space, but whereas a non- clustered index is stored separately from the table, additional storage space is required.

**Unique Index:-**

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.

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

Unique Index is not an separate Index type itself. It's a property.

**Syntax:-**

Create Unique Index <Index\_Name> on <Table\_Name>  
(<Column-Name>);

**Drop Index:-**

Drop index<Tble\_Name>.<Index\_Name>;

**or**

Go to **Object Explorer** (Left side) select the **database** Name expand the **Table** folder and Expand the **Index Folder** click on **Index Name** and select the Particular **Index Name** Right click on it select **Delete**.

#### Useful Point:-

1. By default Primary constraints creates a unique clustered index where as a Unique constraint creates a unique non clustered index. These defaults can be changed if you wish to.
2. A unique index can't be created on an existing table, if the table contains duplicate values. To solve this remove the key columns from the index definition or delete or update the duplicate values.
3. In non-clustered indexes it will occupy more disk space.

#### When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

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

### Set Operators

- Set operators are combines the 2 or more query result sets and then produces a single result set based on the operator.
- SQL supports few Set operations to be performed on table data. These are used to get meaningful results from data, under different special conditions.

#### **Conditions:-**

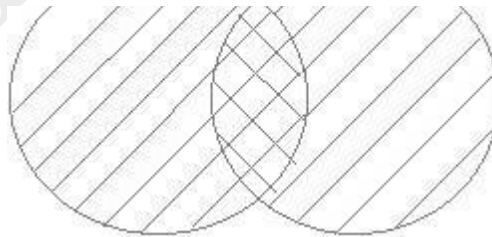
- Number of columns & Order of the columns should be same with in the both queries.
- These columns data types should be compatible.

There are the following 4 set operators in SQL Server:

1. Union
2. Union all
3. Intersect
4. Except

#### **Union:-**

Union is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables.



#### **Note:-**

We have two tables like bellow

**Tbl\_one**

**Tbl\_two**

id	name
1	Sai
2	Aashrith
3	Mahathi

id	name
4	Saanya
2	Aashrith
5	Agasthya

**Syntax:-****Select \* from <Table\_Name> Union Select \* from <Table\_Name>;****E.g:-**

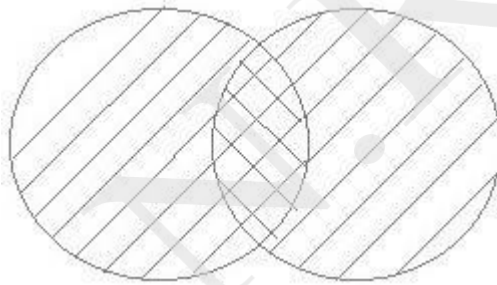
Select \* from tbl\_one union select \* from tbl\_two;

**O/p:** The result table will look like

id	name
1	Sai
2	Aashrith
3	Mahathi
4	Saanya
5	Agasthya

**Union All:-**

This operation is similar to Union. But it also shows the duplicate rows.

**Syntax:-****Select \* from <Table\_Name> Union All Select \* from <Table\_Name>;****E.g:-**

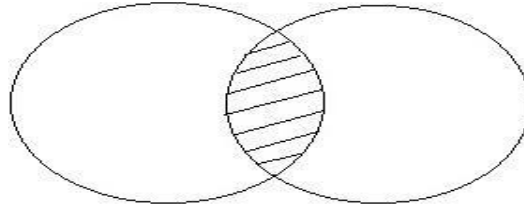
Select \* from tbl\_one union all Select \* from tbl\_two;

**O/p:** The result table will look like

Id	Name
1	Sai
2	Aashrith
3	Mahathi
4	Saanya
2	Aashrith
5	Agasthya

**Intersect:-**

Intersect operation is used to combine two Select statements, but it only returns the records which are common from both Select statements. In case of Intersect the number of columns and datatype must be same.

**Syntax:-**

**Select \* from <Table\_Name> INTERSECT Select \* from <Table\_Name>;**

**E.g:-**

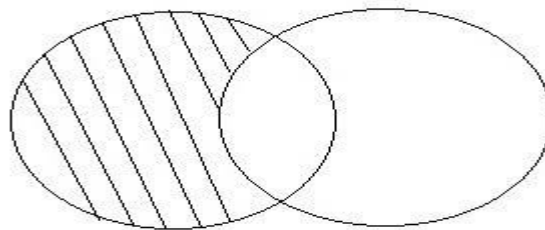
Select \* from tbl\_one INTERSECT Select \* from tbl\_two;

**O/p:** The result table will look like

<u>id</u>	<u>name</u>
2	Aashrith

**EXCEPT:-**

Except operation combines result of two Select statements and return only that result which belongs to first set of result. i.e left hand side table which are not found in right side table.

**Syntax:-**

**Select \* from <Table\_Name> EXCEPT Select \* from <Table\_Name>;**

**E.g:-**

Select \* from tbl\_one EXCEPT Select \* from tbl\_two;

**O/p:** The result table will look like

<u>id</u>	<u>name</u>
1	Sai
3	Mahathi



**IN:-**

The IN operator allows you to specify multiple values in a Where clause. It returns values that match values in a list or sub query. Where IN is a short hand (Extension) for multiple '\_OR' conditions.

**Syntax:-**

**Select <Column\_Names> from <Table\_Name> Where <Column\_Name> IN [(values)];**

E.g:-

1) select \* from tbl\_emp where Eid in (1,2,4,6);

It will return the specified employee details passed in the expression place.

2) select \* from tbl\_emp where Eid not in(1,2,4,6);

It will return the values rest of all values passed in expression.

**Is Null:-**

- Null is a special value that signifies 'no value'.
- Comparing a column to Null using the = operator is undefined.
- Instead, use Where Is Null or Where Is Not Null.

**Syntax:-**

**Select <Column\_Names> from <Table\_Name> where <Column\_Name> Is Null;**

**Select <Column\_Names> from <Table\_Name> where <Column\_Name> Is Not Null;**

E.g:-

Select \* from tbl\_emp where Hire\_Date is Null;

Select \* from tbl\_emp where Hire\_Date is not Null;

### **SUB QUERIES**

- A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the Where clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the Select, Insert, Update, and DELETE statements along with the operators like =, <, >, >=, <=, In, Between etc.
- Subquery contains 2 queries those are Outer Query & Inner Query.
- In Subqueries 1st Inner query will be executed then outer Query will execute. That means outer query is always depends on inner query result.
- Subqueries must be enclosed within parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries are classified into 2 types
  - Non-correlated subqueries
  - Correlated subqueries

#### **Non-correlated subqueries**

Again it classified in to 3 types

- Simple subquery
  - Multiple Subquery
  - Nested Subquery
- A select statement contains two select statements is called as multiple subquery.
  - A select statement contains more than two select statements is called as Nested Subquery.
  - In nested subquery we can write max '255' statements within a single line subquery.

#### **Syntax:-**

**Select <Column\_Names> from <Table\_Name> where [value] in (Select <Column\_Name> from <Table\_Name> Where [condition]);**

---



**Simple Subquery:-****1) Subquery with Select Statement:-**

Select eid,Name,Dept,Salary from tbl\_emp where Salary=(select max(Salary)from tbl\_emp);

o/p :

Eid	Name	Dept	Salary
2	Mahathi	HR	47000

Selecte id,Name,Dept,Salary,Job from tbl\_emp where Eid IN(select Eid from tbl\_emp where salary> 35000);

Eid	Name	Dept	Salary	Job
2	Mahathi	HR	47000	HR Admin

**2)****Subquery with Insert statement:-**

Insert into tbl\_emp\_New select \* from tbl\_emp where Eid IN(select Eid from tbl\_emp);

Consider a table tbl\_emp\_New with similar structure as tbl\_emp table. Now to copy complete tbl\_emp table into tbl\_emp\_New.

Note :-

To create a new table with existing table syntax as follows.

Select \* Into<Destination\_Table>From<Source\_Table>Where 1 = 2;

E.g:-

Select \* Into tbl\_emp\_New From tbl\_emp Where 1 = 2;

**3) Subquery with Update statement:-**

Update tbl\_emp\_New set salary=salary\* 2 where age IN (select age from tbl\_emp where age>= 27 );

**4) Subquery with Delete statement:-**

Delete from tbl\_emp\_New where age IN (Select age from tbl\_emp where age>= 27 );

**Multiple Subquery :-**

```
Select * from tbl_emp where Salary=(select max (salary)
From tbl_emp where Salary<(Select max(Salary)from tbl_emp));
```

It would returns 2<sup>nd</sup> high salary from the table tbl\_emp.

**Nested Subquery:-**

```
Select * from tbl_emp where Salary=(select max(salary)
From tbl_emp where Salary<(Select max(Salary)from tbl_emp where
salary<(Select max(Salary)from tbl_emp)));
```

It would returns 3<sup>rd</sup> high salary from the table tbl\_emp.

**Correlated Subquery:-**

- SQL Correlated Subqueries are used to select data from a table referenced in the outer query.
- In correlated subquery mechanism inner query is always depends on the result of outer query.
- In this type of queries, a table alias must be used to specify which table reference is to be used.
- The alias is the pet name of a table which is brought about by putting directly after the table name in the FROM clause. This is suitable when anybody wants to obtain information from two separate tables.
- Correlated subquery will work on (n-1) mechanism whereas subqueries are working on n+1 mechanism.

**Syntax to find top 10 records:-**

```
Select * from <Table_Name><Alias_Name> where n>
(Select count(<Col_Name>) from <Table_Name ><Alias_Name2>
where <Alias_Name2>. <Col_Name>(>/<)
<Alias_Name1> .<Col_Name>);
```

**Examples:-**

- 1) Select \* from tbl\_empe1 where 2 =  
(select count(salary) from tbl\_empe2  
Where e2.Salary > e1.Salary);  
(OR)

Select top 1 Salary from (select distinct top 3 salary from Tbl\_Emp  
order by Salary desc) s order by salary;

Returns the 3<sup>rd</sup> max salary from the table Tbl\_Emp;

- 2) To get Top 3 highest employee salary list.

Select \* from tbl\_emp E1 where 3 > (select count (Salary) from Tbl\_Emp  
E2 where E2.Salary > E1.Salary) order by Salary desc;

**Note:-**

To find n<sup>th</sup> record ----- (n-1)

To find top n record - - (n >)

**CREATE TABLE Tbl\_Agents**

(

Agent\_Code      CHAR(6) NOT NULL PRIMARY KEY,  
Agent\_Name      Varchar(40),  
Phone\_No        Bigint,  
Country         Varchar(25)

);

Agent_Code	Agent_Name	Phone_No	Country
AG101	Kris	80123376	India
AG102	Murthy	1123455	USA
AG103	Ravi	44123412	UK
AG104	Viswanath	1456788	USA
AG105	Phani	1123788	USA

**CREATE TABLE Tbl\_Customer**

```
(  
    Cust_Code      Varchar(10) NOT NULL PRIMARY KEY,  
    Cust_Name      Varchar(40) NOT NULL,  
    Cust_City      Varchar(35),  
    Cust_Country   Varchar(35) NOT NULL,  
    Phone_No       Bigint NOT NULL,  
    Agent_Code     Char(6) NOT NULL REFERENCES Tbl_Agents,  
    Grade          int  
);
```

Cust_Code	Cust_Name	Cust_City	Cust_Country	PHONE_NO	AGENT_CODE	grade
CS001	Mahesh	Hyderabad	India	40987321	AG101	1
CS002	Ghouse	CA	USA	1987321	AG102	2
CS003	Ramesh	Florida	USA	1973321	AG103	3
CS004	Siva	Hyderabad	India	40456821	AG101	1
CS005	Sekhar	CA	USA	1456821	AG102	2

**Any, All:-**

- Any and All keywords are used with a Where or Having clause.
- Any and All operate on subqueries that return multiple values.
- Any returns true if any of the subquery values meet the condition.
- All returns true if all of the subquery values meet the condition.

**Syntax for ANY:-**

```
Select <Column_Names> from <Table_Name> where  
<Column_Name>[operator] ANY  
(Select <Column_Name> from <Table_Name>  
[where condition]);
```

**E.g:-**

```
Select * from tbl_agents where Agent_Code=any  
(Select Agent_Code from tbl_agents where Country='USA');
```

**Syntax for All:-**

**Select <Column\_Names> from <Table\_Name> where  
<Column\_Name>[operator] ALL  
(select <Column\_Name> from <Table\_Name>  
[Where condition]);**

**E.g:-**

1) select Agent\_code, Agent\_Name from Tbl\_Agents where agent\_Code > All (Select agent\_code from Tbl\_Customer where Cust\_Code='CS001');

2) select eid, name, Salary from Tbl\_Emp where salary > all(select avg(salary) from Tbl\_Emp group by Dept);  
--(This query result from Tbl\_Emp Table)

**SOME:-**

\_\_Some' compare a value to each value in a list or results from a query and evaluate to true if the result of an inner query contains at least one row. \_\_Some' must match at least one row in the subquery and must be preceded by comparison operators. Suppose using greater than ( > ) with \_\_Some' means greater than at least one value.

**Syntax:-       Select <Column\_Names> from <Table\_Name> where  
                  <Column\_Name>[operator] SOME  
                  (select <Column\_Name> from <Table\_Name>  
                  [Where condition]);**

**E.g:-**

1) select eid, name, address from Tbl\_Emp where Eid=some(select Eid from Tbl\_Emp where Address='Ongole');

2) select Agent\_code, Agent\_Name, Country from Tbl\_Agents where agent\_Code= some (Select agent\_code from Tbl\_Customer where Cust\_Country='India');

**EXISTS:-**

- Where Exists tests for the existence of any records in a subquery.
- Exists returns true if the subquery returns one or more records.
- Exists is commonly used with correlated subqueries.

**Syntax:-**

**Select <Column\_Names> from <Table\_Name> where exists (select column-name from <Table\_Name> where [condition])**

**Eg:-**

- 1) Select Agent\_code, Agent\_Name, Country from Tbl\_Agents where exists (Select \* from Tbl\_Customer where grade=3 and Tbl\_Agents.Agent\_Code = Tbl\_Customer.AGENT\_CODE);
- 2) Select Agent\_code, Agent\_Name, Country from Tbl\_Agents where exists (Select \* from Tbl\_Customer where Tbl\_Agents.Agent\_Code = Tbl\_Customer.AGENT\_CODE and Cust\_Country='USA');

### **Synonyms**

- Synonyms are used to create the permanent alias names for the database objects like Tables, Views, Procedures, etc...
- Use the Create statement to create Synonym.
- If we perform the DML operations on the original table then the corresponding synonym also effected and vice versa.
- It will create on the entire table only not possible to create on specific column.
- Synonym are created on a single table only .
- Synonym will become an individual object at the following situation
  - Changing the Base (Original) Table name
  - Dropping the Base table from database.
- On individual synonym we can't perform the DML operations.
- When we change the structure of Base table then the corresponding synonym structure also will change. But we can't change the structure of Synonym individually.

#### **Syntax:-**

Create Synonym <Synonym\_Name> for <Table/Object\_Name>;

E.g:-

Creating a synonym to access the data from a base table Tbl\_Emp

Create synonym Tbl\_Emp\_Syn for Tbl\_emp\_new;

Whenever we execute the above query it will create a synonym on Tbl\_Emp table for accessing the data from the table.

To access the data within synonym:-

Select \* from Tbl\_Emp\_Syn;

Inserting values into Synonym:-

Insert into Tbl\_Emp\_Syn values

(7,'Dvaipaayan',25,'M','IT','Ongole',32000,9875212,'Dvaipaayan.com',null,'Sr.Developer');

----*Inserting values into Table*

Insert into Tbl\_emp\_new values

(8,'Raju',25,'M','HR','Hyderabad',22000,9875212,'Raju.net',null,'Jr.Hr Executive');

Updating values into Synonym:-

Update Tbl\_Emp\_Syn set Address='Hyderabad' where Eid=3;

Updating values into Synonym:-

Delete from Tbl\_Emp\_Syn where Eid=8;

Dropping a Synonym:-

Drop synonym <Synonym\_Name>;

### **Advantages:-**

- Synonyms are used to force hiding the Original Tables/Object's.
- Generally Synonyms are implementing in remoteing environment for accessing the data from one location to another location.
- It will be used to share the objects/ Tables information to multiple programs in the organization.

### **Drawback":-**

- The main drawback of synonym is it does not provide the security to the data of a table.
- We can't create synonyms on particular columns.
- To overcome this we use `_Views`.



### **Views**

- View is an object which is like a table but it's logical / virtual form.
- A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.
- A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view.
- View will not store records in it and will not occupy memory.
- Whenever user performs any operations like Select, Insert, Update or Delete internally the view performs those operations on that table.
- Simply we can say the View will act as an interface between the user & Table.

Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

#### **Types of Views:-**

Users can create 2 types of views on the tables

1. Simple View
2. Complex View

#### **Simple View:-**

- When user create a view on a single Base table that's called Simple View.
- On simple view user can perform DML operations so that Simple View is called as Updatable view.
- We can only insert data in simple view if we have primary key and all not null fields in the view.

#### **Complex View:-**

- Whenever user creates a view on more than one base table then it's called as Complex View.
- On Complex view user can't perform DML operations so that Complex View is called as Not Updatable view.
- We can only update data in complex view. We can't insert data in complex view.

Syntax:-

Create View <View\_Name> as select <Column\_Name's>  
from <Table\_Name> where [condition];

Note:-

You can include multiple Tables in your select statement in very similar way as you use them in normal SQL select query.

E.g on Simple View:-

Creating View:-

Create view Vw\_tbl\_emp as select eid, name, salary  
From Tbl\_Emp; --- (Simple View)

Selecting Data in a View:-

Select \* from Vw\_tbl\_emp;

Note:- Generally View name starts with 'VW' followed by name.

With check option:-

The with check option is a create view statement option. The purpose of the with check option is to ensure that all Update and Inserts satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the Update or Insert returns an error.

E.g:-

Create view Vw\_tbl\_emp as select Eid, Name, Salary from Tbl\_Emp  
where Salary>15000 with check option;

Insert into Vw\_tbl\_emp values (10,'Rama', 15000);  
--- Not allowed to insert.

Insert into Vw\_tbl\_emp values (10,'Rama', 17000);  
---- Allowed to insert.

E.g on complex views:-

Creating a View for accessing the data from 2 tables  
(Tbl\_Agents & Tbl\_Customers)

- 1) Create view vw\_tbl\_Agents\_Customer as select a.agent\_code, a.agent\_Name, c.Cust\_Code, c.cust\_Name, c.Cust\_country from tbl\_agentsa, Tbl\_Customerc where a.agent\_Code=c.Agent\_Code;
- 2) Create view vw\_tbl\_Agents\_Customer\_3 as select a.agent\_code, a.agent\_Name, c.Cust\_Code, c.cust\_Name, c.Cust\_country from tbl\_agentsa join Tbl\_Customerc on a.agent\_Code=c.Agent\_Code;

Note:-

Now we can't perform the DML operations on a view because it was created on multiple tables so that this view is called as complex view.

Whenever we want to restrict the DML operations on the base table than we create a complex view.

Updating a View:

A view can be updated under certain conditions

Update Tbl\_emp\_View set salary=20000 where eid=1;

Delete record from View:-

Delete from Tbl\_emp\_View\_2 where eid=10;

Drop View:-

Drop view<View\_Name>;

**Advantages of Views:-**

- We make Views for security purpose since it restricts the user to view some Columns/Fields of the Table(s).
- Data abstraction since the end user is not aware of all the data present in database table.
- It is used for sharing the data to the multiple users in the organizations.

**Limitations:-**

- You can't create a parameterized view, in other words you can't create a view with a parameter.
- The Select clause may not contain the keyword Distinct, summary functions, set functions, set operators, Order By clause.
- The From clause may not contain multiple tables.
- The Where clause may not contain subqueries.
- The query may not contain Group by or Having.
- Calculated columns may not be updated.
- All Not Null columns from the base table must be included in the view in order for the Insert query to function.

**Differences between Synonyms & Views**

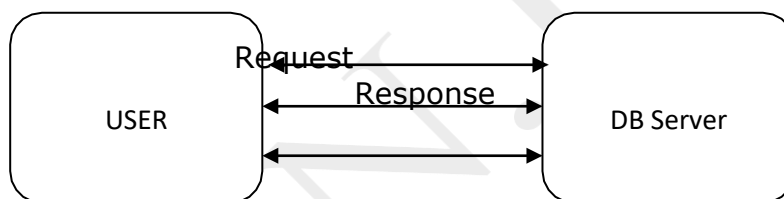
<b>Synonyms</b>	<b>Views</b>
It is alias name of a base table	It is virtual or logical table of base table.
It can create on a single base table only.	It will create multiple base tables at a time.
Synonyms are not providing Hiding/ Restriction facilities on the required Columns.	It will provide hiding facility on the specified Rows/ Columns.
It contains the physical existence and will occupy memory.	It doesn't contain physical existence and will not occupy the memory.

# T-SQL

## Transact Structured Query Language

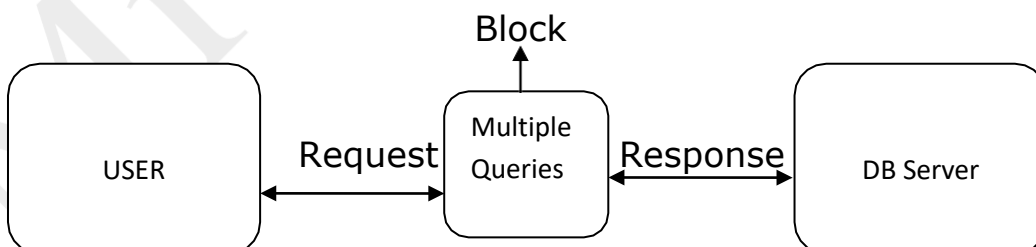
- T-SQL is Procedural language in SQL. Whereas SQL is Non-Procedural language. T-SQL also called as PL/SQL in Oracle.
- The language which supports Conditional & Control statements is called Procedural language.
- SQL doesn't support Conditional & Control statements like if else, multiple if ... while loop statements but these are possible in T-SQL.
- T-SQL will supports only 'while' control statement.
- Declaration of variable is not possible in SQL, but it is possible in T-SQL.
- We cannot implement Exception mechanism in SQL, it is possible T-SQL.
- SQL doesn't provide the re-usability to the queries, T-SQL provides the re-usability.

Query execution process in SQL:-

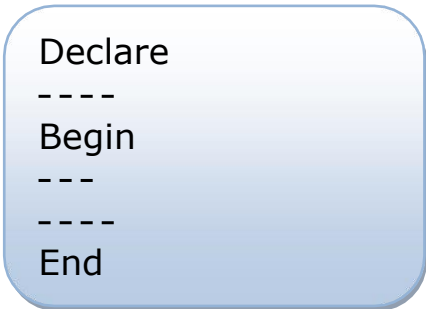


In this case every query request is executed by the database server individually. So that application performance will reduce.

Query execution process in T-SQL:-



In this case the user writing all query statements within a block and this block of statements are executed by the database server as a single query unit. So that the burden of server will reduce & performance also increases.

*Structure of T-SQL Program:-*

```
Declare
----
Begin
---
----
End
```

Block: - Block is nothing but set of statements are executed by the database server as single unit as known as block.

*Declare block:-*

This block is used to declare the variables.

E.g:- @variablename datatype.

*Begin block:-*

This block is used to initialize the values for the variables. In TSQL program if u wants to initialize the value for the variables we have to use set (or) select command.

E.g:- set @ variablename = value

Select @ variablename = value

*End block:-*

This block is used to end the TSQL program.

*What is variable?*

Variable is the name given for a particular memory location where the data is located.

*What is the purpose of variable?*

To identify (or) to access the data.

Cast ():- it is a predefined conversion method which is used to convert from one data type to another data type.

E.g:- cast(<var.name> as <target datatype>)

*// Write a TSQL program to perform arithmetic operations on the given two numbers.*

```
declare
@a int,@b int,@c int
begin
set @a=10
set @b=20
set @c=@a+@b
print'sum is:'+cast(@c as char)
set @c=@a-@b
print'sub is:'+cast(@c as char)
set @c=@a*@b
print'Mul is:'+cast(@c as char)
set @c=@a/@b
print'Div is:'+cast(@c as char)
end
```

O/P : sum is:25  
sub is:15  
Mul is:100  
Div is:4

*// Write a TSQL program to declare First name & last name and display full name.*

```
declare
@fname varchar(20),
@LName varchar(20),
@Fullname varchar(30)
begin
set @fname='sai'
set @LName='Machavaram'
set @Fullname=@fname+@LName
print'Full Name:'+@Fullname
end
```

O/P : Full Name:saiMachavaram

*// Write a TSQL program to find the biggest values among the 2 numbers.*

```
declare
@a int,@b int
begin
set @a=100;
set @b=150
if (@a>@b)
print'a is big number'
else if(@a=@b)
print'both are equal'
else
print'b is big number'
end
```

O/p: b is big number

*// Write a TSQL program to declare sid,sname,sub1,sub2,sub3 and calculate total arks and percentage.*

```
declare
@sid int,@sname varchar(15),
@sub1 int,@sub2 int,@sub3i nt,
@tot int,@Percentage float
begin
set @sid=111
set @sname='Aasthrith'
set @sub1=79
set @sub2=85
set @sub3=92
set @tot=@sub1+@sub2+@sub3
set @Percentage=@tot/3
print'Total is:'+cast(@tot as varchar(20))
print'Percentage is:'+cast(@percentage as varchar(20))
end
```

O/p:      Total is: 256  
        Percentage is: 85



*// Write a TSQL program to find the given number is even or odd.*

```
declare
@i int
begin
set @i=8
if((@i%2)=0)
print 'I is even'
else
print 'I is odd'
end
```

O/p : I is even

*// Write a TSQL program to find the given number is Positive or Negative.*

```
declare
@a int
begin
set @a=-9
if(@a>0)
print 'X is Positive number'
else
print 'X is Negative number'
end
```

O/p : X is Negative number

*// Write a TSQL program to Employee details.*

```
declare
@eid int, @ename varchar(20), @bsal money,
@da money, @hra money, @gross money
begin
set @eid=111
set @ename='Agasthya'
set @bsal=20000
set @da=0.5*@bsal
set @hra=0.4*@bsal
```

```
set @gross=@bsal+@da+@hra
print'EID is:'+cast(@eid as char)
print'ENAME is:'+cast(@ENAME as varchar(20))
print'Bsal is:'+cast(@bsal as char)
print'Da is:'+cast(@da as char)
print'Hra is:'+cast(@hra as char)
print'Gross is:'+cast(@gross as char)
end
```

O/p:-

```
EID is:          111
ENAME is:        Agasthya
Bsal is:         20000.00
Da is:           10000.00
Hra is:           8000.00
Gross is:        38000.00
```

### **While:-**

It will execute the condition repeatedly until the condition becomes `_True_`.  
Whenever the numbers of iterations are not fixed we will go for while loop.  
(Or)

A while loop will check the condition first and then executes the block of Sql Statements within it as long as the condition evaluates to true.

### **Syntax:**

WHILE **Condition**

BEGIN

**Sql Statements**

END

*// Write a TSQL example on While condition.*

```
Declare @i int
Set @i=0
while (@i<10)
begin
set @i=@i+1
print@i
end
```

O/p: 1
2
3
4
5
6
7
8
9
10

**BREAK Statement:-**If a Break statement is executed within a While loop, then it causes the control to go out of the while loop and start executing the first statement immediately after the while loop.

*// Write a TSQL example on While with break statement*

```
Declare @LoopCounter int= 1
while (@LoopCounter<= 4)
begin
print@LoopCounter
if(@LoopCounter= 2)
break
set @LoopCounter=@LoopCounter+ 1
end
print'Statement after while loop'
```

*O/p:*

1

2

*Statement after while loop*

## **Stored Procedure"s**

- A stored procedure is a data base object which contains precompiled SQL queries.
- 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
- You can save the stored procedure with a specific name. And call it just by its name.

Stored Procedures are 2 types

1. System defined stored procedures
2. User defined stored procedures

### **System defined stored procedure:-**

These are created by Microsoft. All the system defined stored procedures always starts with `__sp_``.

E.g:-

<b>System Procedure</b>	<b>Description</b>
sp_rename	It is used to rename an database object like stored procedure, views, table etc.
sp_changeowner	It is used to change the owner of an database object.
sp_help	It provides details on any database object.
sp_helpdb	It provides the details of the databases defined in the Sql Server.
sp_helptext	It provides the text of a stored procedure reside in Sql Server
sp_depends	It provides the details of all database objects that depend on the specific database object.

**User defined stored procedure:-**

The stored procedure that was created by the user depending on the requirement is called as user defined stored procedure.

**Syntax:-**

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

Note:-

1. Use `_create procedure` or `_create proc` statement to create SP.
2. When naming user defined stored procedures, Microsoft recommends not to use `sp_` as a prefix. All system SP are prefixed with `sp_`.
3. This avoids any ambiguity between user defined and system defined and any conflicts.

**Syntax for To Execute the Stored Procedure:-****Option 1: -**

--> Storedprocedure\_name

E.g:- By calling stored procedure name directly like `spGetEmpDetails`

**Option 2: -**

--> **Exec** Storedprocedure\_name

**Option 3: -**

--> **Execute** Storedprocedure\_name

→ Here `spGetEmpDetials` is my stored procedure name.

**Advantages:-**

- Reduce burden on Database & Help in reducing network usage.
- Provides more scalability to an application and also performance will increase.
- Reusable and hence reduce code & security .
- User will get quick response.

**Creating Stored Procedures:-****// Write a Stored Procedure to print Hello.**

```
Create procedure spsai
as
begin
print'Hello Sai Machavaram'
end
```

When you execute the above statements it will create a Stored Procedure as object spsai.

To execute the stored procedure

Execute spsai

(or)

Exec spsai

(or)

spsai

O/p: Hello Sai Machavaram

**// Write a Stored Procedure to perform Addition operation**

```
Create procedure spAddition
```

```
@a int,@b int
```

```
As begin
```

```
Print @a+@b
```

```
end
```

```
----
```

```
Exec spAddition 4,5
```

o/p: 9

**// Write a Stored Procedure to take input as string**

```
Create proc spstring
```

```
@S varchar(30)
```

```
as
```

```
begin
```

```
print'This is :'+@s
```

```
end
```

-----O/p : This is :TSQL

**// create a Stored Procedure to get Employee details in a table.**

```
Create proc spEmpEmployeeDetails
As begin
select * from Tbl_Employee
end
-----
Exec spEmpEmployeeDetails
```

**// create a Procedure to get Employee details in a table by supplying Eid.**

```
Create proc spEmpEmployeeDetailsbyID
@eid int
as
begin
select * from Tbl_Employee where Eid=@eid
end
-----
Exec spEmpEmployeeDetailsbyID 1
O/p:
```

Eid	Ename	Dept_name	Gender	Age	Salary
1	Aashrith	IT	M	32	54000.00

**// create a Procedure to insert Employee details in to Employee table.**

```
Create procedure spEmpInsert
@eid int,@ename varchar(30),@Dept_Name varchar(20), @gender char,
@age int,@salary money
as
begin
insert into Tbl_Employee values
(@eid,@ename,@Dept_Name,@gender,@age,@salary)
end
-----
Execute spEmpInsert 3,'Agasthya','IT','M',29,47590
O/p: (1 row(s) affected)
```

Now if you execute spEmpEmployeeDetails(Employee details SP) there you should find out the 3rd record also.

Exec spempemployeedetails

Eid	Ename	Dept_name	Gender	Age	Salary
1	Aashrith	IT	M	32	54000.00
2	Samhi	HR	F	35	78000.00
3	Agasthya	IT	M	29	47590.00

**// create a Procedure to Delete Employee details from Employee table.**

```
Create procedure spEmpDelete
@eid int
as
begin
delete from Tbl_Employee where eid=@eid
end
```

-----

Execute spEmpDelete 4

When you execute the above Procedure it will delete the record whose Eid is 4 from Tbl\_Employee table.

O/p: (1 row(s) affected)

**// create a Procedure to Update Employee details in Employee table**

```
Create procedure spEmpUpdate
@eid int,@salary money
as
begin
update Tbl_Employee set salary=@salary where eid=@eid
end
```

-----

Execute spEmpUpdate 3,50000

O/p: (1 row(s) affected)



When you execute the above Procedure it will update the salary whose Eid is 3 in Tbl\_Employee table.

**Note:** - Number of parameters and order of parameters, Data type in the SP should be same as no. of values & order of the values at execution time.

**// create a Procedure input the values in to multiple tables.**

```
Create proc spMultiTablesInsert
```

```
@eid int,@ename varchar(30),@Dept_Name varchar(20),@gender char,@age  
int,@salary money,
```

```
@id int,@dname varchar(20),@location varchar(20)
```

```
as
```

```
begin
```

```
insert into Tbl_Employee values
```

```
(@eid,@ename,@Dept_Name,@gender,@age,@salary)
```

```
insert into Tbl_Dept values (@id,@ename,@location)
```

```
end
```

```
-----
```

```
Execute spMultiTablesInsert
```

```
4,'Karthik','Finance','M',31,48920,3,'Finance','Hyderbad'
```

**O/p :-** (1 row(s) affected)

(1 row(s) affected)

**// create a Procedure by using inner join of 2 tables.**

```
Create proc spjoins
```

```
As begin
```

```
select Eid,Name,Dept_Name FROM Emp_tbl INNERJOIN Dept_Tbl on
```

```
Emp_tbl.Eid = Dept_Tbl.id
```

```
end
```

```
-----
```

```
Exec spjoins
```

O/p:

Eid	Name	Dept_Name
1	Aashrith	IT
2	Mahathi	HR
3	Karthik	Sales
4	Saamya	Marketing

### **Altering A Stored Procedure:-**

This is used to modify the previously user defined SP.

#### **// Altering a Procedure.**

Alter procedure spAddition

@a int,@b int,@c int

As begin

Print @a+@b+@c

end

-----

Execute spAddition 10,5,10

**O/p: 25**

### **Stored Procedures types of Parameters:-**

In SQL Server a stored procedure can support 2types of parameters

1. In / Input
2. Out / Output

**Input:** - it is default mode which can be used to accept input values to the stored procedure.

**Output:** - It's used to return the output values through a procedure.

**// create a Procedure for Addition .**

```
Create procedure spAddition3
@a int,@b int,@c int out
As begin
set @c=@a+@b
end
-----
Execute spAddition3 10,5
```

When we execute the above SP we will get an error i.e  
-Procedure or function 'spAddition3' expects parameter '@c', which was not supplied.‖

To overcome this we should follow 3 steps

**Step 1:-** ( Declaring a bind variable for out parameter modes)

Syntax:- Declare @ <bind\_Variable\_Name> [Datatype](size)

E.g:- Declare @a int

**Step 2:-** (Adding bind variable to Procedure)

Syntax :- Execute <Proc\_Name> values..... @<bind\_Variable\_Name> out.

**Step 3:-** (Print bind variable values)

Syntax :- Print @ <bind Variable\_Name>

E.g:- Print @a

**// create a Procedure for Addition ( Above Example)**

```
Create procedure spAddition3
@a int,@b int,@c intout
As begin

Set @c=@a+@b
end
-----
```

Declare @x int

Execute spAddition3 10,30,@x out  
print'Addition is:'+cast(@x as varchar)

**O/P:** Addition is: 40

**Note:-**

If you didn't specify the Out/ Output keyword when executing the stored procedure, the variable will be NULL.

**// create a Procedure which returns password and the length of password for the given userid**

Create a small table with following Columns

```
Create table Tbl_Users  
(  
  User_Name varchar(30) primary key,  
  Password varchar(15)  
)
```

User_Name	password
Aashrith	Ash@123
Karthik	Karthik@123
Saamya	Saamya@123
Samhi	Samhi@123

Create proc spPswdLen

```
@User_Name varchar(20), @password varchar(15)out,  
@lenngth int out
```

```
As begin
```

```
Select @password=password, @lenngth=len(password) from Tbl_Users where  
User_Name=@User_Name
```

```
end
```

--- press F5/ Click Excute button

```
Declare @x varchar(20),@y int
Exec spPswdLen'Aashrith',@x out,@y out
print'Password is:'+'@x
print'Lenght of Password is:'+'@y as varchar)
```

**O/P:**

Password is:Ash@123  
Lenght of Password is:7

**// create a Procedure which will verify User name & Password if both are valid returns "Valid User" else returns "Invalid Credentials".**

```
Create proc spCheckUserDetails1
@User_Name varchar(20),@password varchar(15),
@Status varchar(20)out
As begin

if (Select count(*)from Tbl_Users where User_Name =@User_Name and
password=@password)>0
set@Status='Valid User'
else
set@Status='Invalid Credentials'
end
```

--- press F5/ Click Excute button

```
Declare @x varchar(20)
Exec spCheckUserDetails1 'Aashrith','Ash@123',@x out
Print @x
```

**O/p:** Valid User

**Rename SP:-**

```
sp_rename 'old_procedure_name', 'new_procedure_name'
```

(OR)

```
Exec sp_rename 'SalesByCustomer', 'NewSalesByCustomer';  
go
```

**Drop SP:-**

```
Drop procedure <stored_procedure_name>
```

## Functions

- Functions are nothing but set of statements which will perform some operations and returns a value to the user.
- Which will similar to the Stored Procedures

Functions are classified into 2 types

- System defined functions (Inbuilt) (Discussed in SQL)
- User defined functions

### **User Defined Functions(UDF):-**

The function that was created depending on the user requirement.

User Defined Functions play an important role in SQL Server. User Defined functions can be used to perform a complex logic, can accept parameters and return data. Many a times we have to write complex logic which cannot be written using a single query. In such scenarios, UDFs play an important role. For example, we can call user defined function in a where clause or use a user defined function in a JOIN [Where UDF returns a result set]

SQL Server supports two types of User Defined Functions

#### **1) Scalar Functions**

- The function which returns a Scalar/Single value.

**Syntax:-** *Create function <Function\_Name> [Parameter(s) Datatype(s)]  
Return datatype[size]  
As begin  
--Declare the variables  
-- initialize the variables  
Return value/variable/expression  
End*

A Create function statement is used to create a Scalar-valued function. The name of the function should not be more than 128 characters. It is not a rule but it is conventional that the name of the function should begin with the prefix **fn.**

Up to 1024 input parameters can be defined for Scalar-valued functions. A Scalar-valued function however cannot contain an output parameter. The value is returned by a Scalar-valued function using the **RETURNS** clause.

Scalar functions may or may not have parameters, but always return a single (Scalar) value. The returned value can be of any data type, except \_Text, Ntext, Image and timestamp.

**Syntax to call a Scalar Function:-**

Select dbo.<Function\_Name>(value(s))

Dbo:- Database owner

**Examples:-****// create a function which returns Addition of 2 numbers**

Create function fn\_addition(@a int,@b int)

Returns int

As begin

Declare @c int

Set @c=@a+@b

Return @c

end

-----

Select dbo.fn\_addition(15,35)

**o/p : 50**

**// create a function which returns the cube of given number**

Create function fn\_cube(@c int)

Returns int

As begin

Declare @d int

Set @d=@c\*@c\*@c

Return @d

end

-----

Select dbo.fn\_cube(6)

**o/p : 216**



**// create a function which will take employee id & returns employee salary from the Tbl\_emp table**

Create function fn\_empsal(@eid int)

Returns money

As begin

Declare @empsal money

Select @empsal=Salary from Tbl\_Emp where eid=@eid

Return @Empsal

end

-----

Select dbo.fn\_empsal(4)

O/p: 35000.00

## **2) Table Valued Functions**

- This function will take the input from the user and return multiple rows from the table.
- The return type of table valued function is Table.
- In this function we are not having 'as begin' and 'end' block

Table Valued Functions can be written as

- Inline Table
- Multi-statement Table

**Syntax:** - Create function <Function\_Name> [Parameter(s) Datatype(s)]  
Return table  
As  
Return (select query)

### **Syntax for calling a Table value Function:-**

Select \* / <Column\_list> from <Function\_Name>(Values)

### **// Create a table like bellow**

Create table tbl\_Customers

(

C\_id int,

C\_name varchar(20),

C\_State varchar(20)

)

```
Insert into tbl_Customers values (111,'Aasthrith','Andhrapradesh')
```

```
Insert into tbl_Customers values(222,'Samhi','Tamilnadu')
```

```
Insert into tbl_Customers values  
(333,'Agasthya','Madhya Pardesh')
```

```
Select * from tbl_Customers
```

C_id	C_name	C_Satate
111	Aasthrith	Andhrapradesh
222	Samhi	Tamilnadu
333	Agasthya	Madhya Pardesh

### In line Table-Valued user defined Function:-

An inline table-valued function returns a variable of data type table whose value is derived from a single SELECT statement. Since the return value is derived from the SELECT statement, there is no BEGIN/END block needed in the CREATE FUNCTION statement. There is also no need to specify the table variable name (or column definitions for the table variable) because the structure of the returned value is generated from the columns that compose the SELECT statement. Because the results are a function of the columns referenced in the SELECT, no duplicate column names are allowed and all derived columns must have an associated alias.

**//In this Example which accepts city as the input parameter and returns customer code & Name of all belongs to the input city.**

```
Create function fn_GetCustdetailsbyState  
(@C_State varchar(20))  
RETURNS table  
AS  
RETURN (  
SELECT C_id, c_Name  
FROM tbl_Customers  
WHERE C_State=@C_State  
)  
GO
```

-----

Select \* from fn\_GetCustdetailsbyState('Andhrapradesh')

O/P:

C_id	c_Name
111	Aasthrith

### **Multi – statement Table-Valued user defined Function:**

A Multi-Statement Table-Valued user-defined function returns a table. It can have one or more than one T-Sql statement. Within the create function command you must define the table structure that is being returned. After creating this type of user-defined function, we can use it in the FROM clause of a T-SQL command unlike the behavior found when using a stored procedure which can also return record sets.

**//In this Example which accepts city as the input parameter and returns customer code & Name of all belongs to the input city. If for the input city there are no customers then this UDF will return a record with no customer values found**

Create function fn\_GetCustdetailsbyStateMS

(@C\_State varchar(20))

Returns @tbl\_Customers1 table(c\_id int,c\_Name varchar(20))

As begin

Insert into @tbl\_Customers1

Select c\_id,C\_Name from tbl\_Customers where C\_State=@C\_State

If @@rowcount=0

begin

insert into @tbl\_Customers1

values ('','No Records found')

end

return

end

go

-----

Select \* from fn\_GetCustdetailsbyStateMS('Karnataka')

---

c_id	c_Name
0	No Records found

**Other Examples on In line table UDF :-**

**// create a function which will take employee id & returns employee details from the table Tbl\_emp .**

Create function fn\_EmpDetails(@eid int)

Returns table

as

return (Select \* from tbl\_emp where Eid=@eid)

----- press F5 or Click Execute

Select \* from fn\_EmpDetails(2)

O/p:

Ei d	Name	Ag e	Gend er	Dep t	Address	Salary	Cell_No	EMail	Hire_Dat e	Job
2	Mahat hi	32	F	HR	Hyderab ad	11750. 00	68784565 20	mahathi_2@gmail .com	2012-10- 23	HR Admi n

**// create a function which will take Dept name & returns the list of employee details who are working the given Dept from the table Tbl\_emp .**

Create function fn\_DeptEmpDetails(@Dept varchar(20))

Returns table

as

return (Select \* from tbl\_emp where Dept=@Dept)

-----

Select \* from dbo.fn\_DeptEmpDetails('it')

**O/P:**

Eid	Name	Age	Gender	Dept	Address	Salary	Cell_No	Email	Hire_Date	Job
1	Aashrith	28	M	IT	Ongole	20000.00	9966640779	parasara189@gmail.com	2014-10-06	SR. Developer
6	Agasthya	29	M	IT	Ongole	32000.00	5656878420	agasthya_6@ms.net	NULL	SR. Developer
7	Dvaipaayan	25	M	IT	Ongole	25000.00	875468	Dvaipaayan.com	NULL	Sr.Developer

// create a function which will accept Eid & returns the required employee details from Emp\_Tbl & Dept\_Tbl tables.

Create function fn\_JoinsEmpDetails(@eid int)

Returns table

as

return (select e.Eid,e.Name,d.Dept\_Name FROM Emp\_tble INNER JOIN Dept\_Tbl d on e.Eid=d.id where e.Eid=@eid)

-----  
Select \* from dbo.fn\_JoinsEmpDetails(1)

Eid	Name	Dept_Name
1	Aashrith	IT

**Syntax to drop a function:-**

Drop function<Function\_Name>

Drop function fn\_JoinsEmpDetails

**Limitations and Restrictions**

- User-defined functions cannot be used to perform actions that modify (Insert/ Update/Delete) the database state.
- Alter Function cannot be used to perform any of the following actions:
  - Change a scalar-valued function to a table-valued function, or vice versa.
  - Change an inline function to a multi statement function, or vice versa.

**Deterministic**

A deterministic User defined function (UDF) always returns the same result with the same set of input parameters. Some examples of deterministic functions are the system functions MONTH(), YEAR(), and ISNULL().

**Non-deterministic**

A non-deterministic UDF is can potentially return a different value each time it is called with the same set of input parameters. Some examples of non-deterministic functions are the system functions GETDATE(), NEWID(), and @@CONNECTIONS.

**Differences between Scalar value & Table valued Functions**

Scalar Valued	Table Valued
It will return only one value	It will return one value (or) multiple rows from table
Returns type is Datatype	Return type is Table
It will have as begin & end block	It will not have as begin and end block
Syntax for calling this function Select dbo.function_name(values)	Syntax for calling this function Select */Columns from dbo.function_name(values)

**Differences between Stored Procedure & Functions**

<b>Stored Procedure</b>	<b>Functions</b>
It is a set of pre compiled SQL statements which will executes when we call SP	Function is Re-Usable piece of code which will gets executed when we call it.
It will create Execution plan	It will not having execution plan
It will return always integer data type	It will return any data type
Procedure never return a value if it will return a value use OUT	A function must return a value RETUTRN
It will support exception handling	It will not support exception handling
It will accept input & O/p parameters	It will accept only input parameters
It may/ may not accept parameters	It must accept at least 1 parameters
It will execute by EXEC command	It will execute by Select command
We can call function in stored procedure	We can't call stored procedure in function
We can call one stored procedure in another stored procedure	We can call one function in another function.

## **TCL Commands**

Transaction Control Language (TCL) commands are used to manage transactions in database. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

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 transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

SQL Server will support 2 types of transactions those are

1. **Implicit Transactions:** - These transactions are performed by the system by default. These are also called Auto Commit transactions.

E.g:- DDL & DML Operations

Note:- By default the DML operations are Auto commit operations in SQL Server.

Auto Commit:- System commit the DML operations which were performed by the user on the table automatically. It's called Autocomit.

2. **Explicit Transactions:-** These transactions are performed by the user by using TCL commands.

The following TCL commands used to control transactions:

➤ **Begintransaction:**

Begin transaction command is used to start the transaction.

**Syntax:** *Begin transaction*  
{SQL statements}



➤ **Commit:**

Commit command is used to permanently save any transaction into database. The commit command saves all transactions (DML) to the database since the last Commit or Rollback command.

If you do not write commit then your data will be restored into its previous condition.

**Syntax:** *Begin Transaction*

{SQL statements}

*commit;*

➤ **Rollback:**

This command restores the database to last committed state / Start point. It is also use with savepoint command to jump to a savepoint in a transaction. But remember you can't rollback the data once you write commit.

**Syntax:-** *Begin Transaction*

*rollback / rollback to <savepoint\_Name>(if save point assigned)*

➤ **Savepoint / savetransaction :**

A Savepoint is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction. It is used for dividing / breaking transaction into multiple units. So that user can rollback a transaction up to a location.

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

**Syntax:-** *Begin Transaction*

*Save transaction <savetransaction\_Name>*

*{SQL statements}*

## **Examples on TCL commands:-**

Let's create a Table like bellow.

```
Create table Tbl_students(  
S_Id int primary key,  
S_Name varchar(20),  
S_Course varchar(15),  
S_Address varchar(20),  
Course_fee money  
)
```

Begin transaction //transaction begins

Insert into Tbl\_students values

(111,'Vivaan','Dot Net','Hyderabad',25000)

Insert into Tbl\_students values (222,'Aashrith','Java','Ongole',30000)

Insert into Tbl\_students values (333,'Samhi','PHP','Chennai',25000)

Insert into Tbl\_students values (444,'Agasthya','Testing','Nagpur',30000)

S_Id	S_Name	S_Course	S_Address	Course_fee
111	Vivaan	Dot Net	Hyderabad	25000.00
222	Aashrith	Java	Ongole	30000.00
333	Samhi	PHP	Chennai	25000.00
444	Agasthya	Testing	Nagpur	30000.00

In the above transaction if user want's rollback the data he can, because those transactions are not committed.

Begin transaction  
rollback

With this statement we will rollback the data and table will be empty.

Begin transaction //transaction begins

Insert into Tbl\_students values

(111,'Vivaan','Dot Net','Hyderbad',25000)

Insert into Tbl\_students values (222,'Aashrith','Java','Ongole',30000)

Insert into Tbl\_students values (333,'Samhi','PHP','Chennai',25000)

Insert into Tbl\_students values (444,'Agasthya','Testing','Nagpur',30000)

commit; // Transaction Committed

In the above transaction user can't rollback DML operations, because these operations are committed within the transaction.

### -- Another example

Begin transaction

Update Tbl\_students set S\_Address='Secunderabad' where s\_id=111

Insert into Tbl\_students values (555,'Rakesh','Testing','Mumbai',28000)

If you want to roll back the above transactions for that you should write syntax like below

Begin transaction

rollback ;

In the above transaction user can rollback the DML statements because those are not committed.

Begin transaction

Insert into Tbl\_students values (555,'Rakesh','Testing','Mumbai',28000);

Update Tbl\_students set S\_Course='Dot Net' where s\_id=222;

in the above transactions user can Rollback the all DML statements because those statements are not committed.

Begin transaction

Delete from Tbl\_students where s\_id=555

Save transaction sp1

Delete from Tbl\_students where s\_id=444

in the above transaction user was deleted 2 records from the table, but if he wants roll the record back to save the transaction point user need to write the syntax like below.

Begin transaction

Rollback transaction sp1

With this process user will get the record of S\_id i.e 444 will be back.

## DCL Commands

These are Data Control Language Commands used to enforce security in a multi-user database environment.

DCL defines two commands,

- Grant
- Revoke

### **Creating a new user in SQL Server:-**

- Login to SQL Server with Windows Authentication
- Click on 'New query'
- Write the below syntax to create a new user

#### **Syntax:-**

Create login<User\_Name>with password=<'Password'>;

E.g:-

Create login saim with password='sai123';

- With this process it will create a new user with login **saim** & password **sai123**.
- To perform any operation like creating Table, etc., we have to grant the privileges to that user.

#### **Grant : -**

SQL Grant is a command used to provide access or privileges on the database objects to the users.

#### **Syntax:-** Grant <privilege\_name>

ON <object\_name >

TO {user\_name |PUBLIC |role\_name}

[WITH GRANT OPTION];

- **privilege\_name** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- **object\_name** is the name of a database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- **user\_name** is the name of the user to whom an access right is being granted.
- **PUBLIC** is used to grant access rights to all users.
- **ROLES** are a set of privileges grouped together.
- **WITH GRANT OPTION** - allows a user to grant access rights to other users.

**Revoke : -**

The REVOKE command removes user access rights or privileges to the database objects.

Take back permissions from user.

**Syntax:-**

```
Revoke< privilege_Name>  
on <object_Name>  
from{userName/public/roleName}  
To take back Permissions
```

**Note:-**

In real time DCL Commands (Grant & Revoke) will control by only DBA Team. Normal user doesn't have access to these commands.

## Cursors

- Cursor is memory location for storing data bases tables.
- Cursors is temporary work area allotted to the client at server when a select statement is executed
- A cursor contains information of select statement and information of data accessed by it.
- A cursor can hold more than one row but can process only one row at a time.
- The set of rows the cursor holds is called as result set.

Types of cursors:-

1. Implicit Cursor
2. Explicit Cursor

### **1. Implicit Cursor:-**

The cursors which will be created by SQL by default when select statement executes.

Eg:-

// Write a query to display Name whose Eid is 1

Declare @ename varchar(20)

begin

select Name into EmpName from Tbl\_Emp where eid=1

end

Goto → Object Explorer → Databases→ Tables→EmpName

Select \* from EmpName

O/P: Name

Aashrith

- Here by `_select into` clause is used to fetch the data from the table and store in SQL memory area.
- By using this cursor we can fetch only one value from the table.

**2. Explicit Cursor :-**

- The cursor which is created by the user is called as Explicit Cursor.
- By using this we can fetch multiple rows from the table & store in SQL memory area.
- Whenever we want to go for `__record` by record manipulation we will use this.

**Steps to create Explicit Cursor:-****1. Declare the cursor**

Here we will define a cursor

Syntax: `Declare <Cursor _ Name> cursor for (Select column(s) statement)`

**2. Open the cursor**

When we open a cursor it will internally execute the select statement that is associated with the cursor declaration and load the data into cursor.

Syntax: `open <Cursor_Name>`

**3. Fetch the data from the cursor**

In this process when cursor is opened, rows can be fetched from the cursor one by one or in a block to do data manipulation.

Syntax:

`Fetch First / Last / Next / Prior / Absolute n / Relative n from <Cursor_Name> [into <Variables>]`

**4. Close the cursor**

After data manipulation, we should close the cursor explicitly.

Syntax: `Close <Cursor_Name>`

**5. Deallocate the cursor**

Finally, we need to delete the cursor definition and released all the system resources associated with the cursor.

Syntax: `deallocate <Cursor _Name> //` --after deallocation it can't be reopen

**FetchingMethods:-**

**First:** - Fetching the 1<sup>st</sup> record from the table.

**Next:** - Fetching the records in forward directional (From current record to next record) from the table.

**Last:** - Fetching the last record from the table.

**Prior:** - Fetching the records in backward directional (From current record to previous record) from the table.

**Absolute „n“:-** Fetching the exact position record, ' n \_represents position of the record.

**Relative „n“:-** Fetch the row that is after / prior to the current row, \_n \_represents position of the record.

Note:- By default cursors are forward only and it will support only one method i.e \_ Next`.

// Write a cursor prog to Display Name & Salary from Tbl\_Emp table by using cursor variables.

Declare cur\_emp cursor for select name,Salary from Tbl\_Emp

Declare @name varchar(20),@salary money

Open cur\_emp

Fetch next from cur\_emp into @name,@salary

While @@FETCH\_STATUS=0

begin

print'Salary of : '+@name+' is '+cast(@Salaryasvarchar(20))

fetch next from cur\_emp into@name,@salary

end

close cur\_emp

deallocate cur\_emp

O/P: Salary of: Aashrith is: 20000.00

Salary of: Mahathi is: 11750.00

Salary of: Karthik is: 33000.00

Salary of: Saamya is: 35000.00

Salary of: Saanya is: 36000.00

Salary of: Agasthya is: 32000.00

Salary of: Dvaipaayan is: 25000.00



// Write a cursor prog to fetch employee records row by row manner from Tbl\_Emp table without using cursor variables .

Declare cur\_emp cursor for select \* from Tbl\_Emp

Open cur\_emp

Fetch next from cur\_emp

While @@FETCH\_STATUS=0

begin

fetch next from cur\_emp

end

close cur\_emp

deallocate cur\_emp

O/p:

Ei d	Name	Ag e	Gende r	Dep t	Addres s	Salar y	Cell_No	EMail	Hire_Date	JO b
1	Aashrit h	28	M	IT	Ongole	2000 0	996664077 9	parasara189@gmail.c om	10/6/2014	SR. Developer

Ei d	Name	Ag e	Gende r	Dep t	Address	Salar y	Cell_No	EMail	Hire_Date	JO b
2	Mahath i	32	F	HR	Hyderaba d	1175 0	687845652 0	mahathi_2@gmail.c om	10/23/2012	HR Admin

Ei d	Name	Ag e	Gende r	Dep t	Addres s	Salar y	Cell_No	EMail	Hire_Date	JO b
3	Karthik	26	M	DBA	Mumba i	3300 0	845687652 0	karthik_3@ymail.co m	9/18/2013	Sys Admin

Like this it will display the entire records row by row.....

```
// Write a cursor prog to update the Employee salary on the following criteria
If Department is IT increment 20 %( 0.2%)
If Department is HR increment 15 %( 0.15%)
If other Departments increments 12% (0.12%)
Declare cur_emp cursor for select eid,dept from Tbl_Emp
Declare @Eid int,@Dept varchar(20)
Open cur_emp
Fetch next from cur_emp into @Eid,@Dept
While @@FETCH_STATUS=0
begin
if @Dept='IT'
update Tbl_Emp set salary=(salary*0.2)+salary where eid=@Eid
else if @Dept='HR'
update Tbl_Emp set salary=(salary*0.15)+salary where eid=@Eid
else
update Tbl_Emp set salary=(salary*0.12)+salary where eid=@Eid
fetch next from cur_emp into @Eid,@Dept
end
close cur_emp
deallocate cur_emp
```

This will reflect Department wise increment as mentioned above criteria in the table Tbl\_Emp

### **Forward only & Scroll cursors**

If a cursor is defined as forward only it allows you to navigate only to the next records in sequential order. Moreover it supports only fetch `__Next` method (one – by-one)

Whereas a Scroll cursor allows you to navigate / fetch top –bottom or bottom to top and also it will support all fetching methods.

// Write a cursor prog to fetch the records last to first manner.

Declare cur\_emp cursor scroll for select \*from Tbl\_Emp

Open cur\_emp

Fetch last from cur\_emp

While @@FETCH\_STATUS=0

begin

fetch prior from cur\_emp

end

close cur\_emp

deallocate cur\_emp

O/P:

Eid	Name	Age	Gender	Dept	Address	Salary	Cell_No	EEmail	Hire_Date	JOB
7	Dvaipaayan	25	M	IT	Ongole	7200	875468	Dvaipaayan.com	NULL	Sr.Developer

Like this it will fetch all the record from last to first row by row

// Write a cursor prog to fetch the every 2<sup>nd</sup> position (odd) records from the table.

Declare cur\_emp cursor scroll for select eid,Name from Tbl\_Emp

Declare @Eid int,@Name varchar(20)

Open cur\_emp

Fetch first from cur\_emp into @Eid,@name

While @@FETCH\_STATUS=0

begin

print 'The Employee :-'+@Name+' Id is:-'+cast(@eid as varchar)

fetch relative 2 from cur\_emp into @Eid,@Name

end

close cur\_emp

deallocate cur\_emp

O/P: The Employee :-Aashrith Id is:-1

The Employee :-Karthik Id is:-3

The Employee :-Saanya Id is:-5

The Employee :-Dvaipaayan Id is:-7

// Write a cursor prog to fetch the even number position records from the table.

```
Declare cur_emp cursor scroll for select eid,Name from Tbl_Emp
declare @Eid int,@Name varchar(20)
open cur_emp
fetch absolute 2 from cur_emp into @Eid,@name
while @@FETCH_STATUS=0
begin

print 'The Employee : '+@Name+' Id is: '+cast(@eid as varchar)

fetch relative 2 from cur_emp into @Eid,@Name
end
close cur_emp
deallocate cur_emp
```

O/P:

```
The Employee : Mahathi Id is: 2
The Employee : Saamya Id is: 4
The Employee : Agasthya Id is: 6
```

// Write a cursor prog to update 1<sup>st</sup> record salary as 34500 & delete the last record from the table.

```
Declare cur_emp cursor scroll for(select Eid from Tbl_Emp)
Declare @eid int
Open cur_emp
Fetch first from cur_emp into @eid
Update Tbl_Emp set Salary=34500 where eid=@eid
Fetch last from cur_emp into @eid
Delete from Tbl_Emp where eid=@eid
Close cur_emp
Deallocate cur_emp
```

O/P:- When execute the above cursor it will update 1<sup>st</sup> employee salary to 34500 and also it will delete the employee details from the table who places in last position .

// Write a cursor prog to implement all the fetching methods.

```
Declare cur_emp cursor scroll for(select Name,Salary from Tbl_Emp)
Declare @Name varchar(20),@salary money
Open cur_emp
```

```
Fetch last from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast(@salary as varchar)
```

```
fetch prior from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast(@salary as varchar)
```

```
fetch absolute 4 from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast (@salary as varchar)
```

```
fetch relative-1 from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast(@salary as varchar)
```

```
fetch first from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast(@salary as varchar)
```

```
fetch Next from cur_emp into @Name,@salary
print'Employee : '+@Name+' Salary is: '+cast(@salary as varchar)
```

```
close cur_emp
deallocate cur_emp
```

O/P:-

```
Employee : Kalyani Salary is: 40000.00
Employee : Vivaan Salary is: 25678.00
Employee : Saamya Salary is: 1041.47
Employee : Karthik Salary is: 596.09
Employee : Aashri thSalary is: 25400.00
Employee : Mahath iSalary is: 34500.00
```

**Note:-**

**If there is ever a need to process the rows, on row-by-row basis, then cursors are your choice. Cursors are very bad for performance, and should be avoided always. Most of the time, cursors can be very easily replaced using joins.**

**Replacing cursors using joins**

Create 2 tables like bellow

1. Tbl\_Products
2. Tbl\_Product\_Sales

**Syntax for Tbl\_Products:-**

```
Create table Tbl_Products
(
  Id int constraint Prod_pk primary key,
  Name Varchar(20),
  Discription varchar(20)
)
```

**Syntax for Tbl\_Product\_Sales:-**

```
Create table Tbl_Prodcut_sales
(
  Id int constraint PS_id_pk PRIMARY KEY,
  Product_id intconstraint Pr_id_fk references Tbl_Products(id),
  unit_price money,
  Qty_saled int
)
```

**Tbl\_Products**

id	Name	Discription
111	Product-4	Prod_Description
123	Product-1	Prod_Description
222	Product-5	Prod_Description
333	Product-6	Prod_Description
456	Product-2	Prod_Description

**Tbl\_Prodcut\_sales**

id	Product_id	unit_price	Qty_saled
1	111	2000.00	18
2	111	4000.00	22
3	222	5000.00	23
4	333	9000.00	65
5	456	12000.00	15

//If you want to update particular product's price without using cursors by using joins

Update Tbl\_Prodcut\_sales

Set unit\_price=

Case

When name='Product-1' then 78000

When name='Product-2' then 65000

When name='Product-3' then 74000

When name like 'Product-6%' then 50000

else

unit\_price

End

From Tbl\_Prodcut\_sales join Tbl\_Products on

Tbl\_Products.id=Tbl\_Prodcut\_sales.Product\_id where

(Name='Product-1' or name='Product-2' or name='Product-3' or name like 'Product-6%')

O/P: (4 row(s) affected)

Those records will update in Tbl\_Prodcut\_sales table.

**CASE:-**

In SQL Server (Transact-SQL), the CASE statement has the functionality of an IF-THEN-ELSE statement. You can use the CASE statement within a SQL statement.

CASE expression

```
WHEN value_1 THEN result_1
WHEN value_2 THEN result_2
...
WHEN value_n THEN result_n
```

ELSE result

END

CASE

```
WHEN condition_1 THEN result_1
WHEN condition_2 THEN result_2
...
WHEN condition_n THEN result_n
```

ELSE result

END

**Note**

- If no value/condition is found to be TRUE, then the CASE statement will return the value in the ELSE clause.
- If the ELSE clause is omitted and no condition is found to be true, then the CASE statement will return NULL.
- Conditions are evaluated in the order listed. Once a condition is found to be true, the CASE statement will return the result and not evaluate the conditions any further.
- You cannot use the CASE statement to control program flow, instead, use loops and conditional statements.



## **Triggers**

Triggers are database object. Basically these are special type of stored procedure that are automatically fired/executed when a DDL or DML command statement related with the trigger is executed. Triggers are used to assess/evaluate data before or after data modification using DDL and DML statements. These are also used to preserve data integrity, to control server operations, to audit a server and to implement business logic or business rule.

Triggers are 3 types, they are:

1. DML Triggers
2. DDL Trigger
3. Logon Triggers

### **DML Triggers**

- DML triggers are fired automatically in response to DML events (Insert, Update & Delete)

DML Triggers can be again classified into 2types

1. After Triggers ( Sometimes called as FOR triggers)
2. Instead of triggers

**After Triggers:** Fires after the triggering action. The Insert, Update & Delete statements, causes an after trigger to fire after the respective statements complete execution.

**Example:** If you insert record/row in a table then the trigger related/associated with the insert event on this table will fire only after the row passes all the constraints, like as primary key constraint, and some rules. If the record/row insertion fails, SQL Server will not fire the After Trigger.

**Instead of triggers:** fires instead of the triggering action. The Insert, Update, and Delete statements, causes an instead of trigger to fire Instead of the respective statement execution.

**Example:** If you insert record/row in a table then the trigger related/associated with the insert event on this table will fire before the row passes all the constraints, such as primary key constraint and some rules. If the record/row insertion fails, SQL Server will fire the Instead of Trigger.

---

**Note:-**

- We cannot call the trigger by explicitly because whenever the user performs DML & DDL operations then automatically the triggers are invoked.
- Triggers are parameter less objects.

**Syntax to create DML Trigger**

```
Create Trigger <trigger_name>  
ON { table_name | view_name }  
{ FOR | AFTER | INSTEAD OF }  
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }  
AS { sql_statement [ ; ] }  
GO
```

*Ttrigger\_Name:*

This is the name of the trigger. It should conform to the rules for identifiers in Sql Server.

*Table|View :*

This is the table/view on which the trigger is to be created.

*FOR/AFTER*

FOR/AFTER specifies that the trigger is After Trigger. AFTER is the default, if FOR is the only keyword specified. AFTER triggers cannot be defined on views.

*INSTEAD OF:*

INSTEAD OF specifies that the trigger is Instead Of Trigger.

*CREATE|ALTER|DROP|INSERT|UPDATE|DELETE*

These keywords specify on which action the trigger should be fired. One of these keywords or any combination of these keywords in any order can be used.

**AS:-**After this we specify the actions and condition that the trigger performs.

**sql\_statement:-**These are the trigger conditions and actions. The trigger actions specified in the T-SQL statements.

**Create a table to perform trigger operations...**

Create Table Tbl\_Persons

```
(  
ID Int Identity,  
Name Varchar(100),  
Sal Decimal (10,2)  
)
```

*//Create a trigger that to display a message that record is inserted successfully.*

Create trigger tr\_Tbl\_Persons\_insert on Tbl\_Persons

After insert

as

begin

print'Record is Inserted successfully'

end

===== Execute the trigger ===== it will say \_Command(s)  
completed successfully'=====

Insert into Tbl\_Persons values('Aashrith',45642);

=====

O/P:-

Record is Inserted successfully

(1 row(s) affected)

*//Create a trigger that to display Persons details after inserting the record.*

Create trigger tr\_Tbl\_Persons\_Display on Tbl\_Persons

After insert

as

begin

select \* from Tbl\_Persons

end

===== Execute the trigger ===== it will say \_Command(s)  
completed successfully'=====

Insert into Tbl\_Persons values ('Mahathi',47564);

When we insert a record trigger will fire and display all the persons table details as O/p...

**Note:-**

- Whenever we working with triggers two tables will be created internally with Inserted & Deleted.
- Inserted table will be created after performing inserting operation
- Deleted table will be created after performing deleting operation.
- These two tables are „**Magic Tables**“.

**Magic Tables**

- Magic tables are invisible tables or virtual tables.
- You can see them only with the help of triggers in SQL server.
- Magic tables allow you to hold Inserted, updated & deleted values during DML Operations.
- These are two types (--above mentioned in Note points)

**Example on Inserted Magic Table:-**

Create trigger tr\_Tbl\_inserted on Tbl\_Persons

For insert

as

begin

select \* from inserted

end

===== Execute the trigger ===== it will say \_Command(s) completed successfully' =====

Insert into Tbl\_Persons values ('Swami',36000)

When we insert a record into Tbl\_Persons it will show the value in inserted magic table (Holding). After it will transfer to Tbl\_Persons table.

**Example on Deleted Magic Table:-**

Alter trigger tr\_Tbl\_deleted on Tbl\_Persons

For delete

as

begin

select \* from deleted

end

===== Execute the trigger=====

Delete from Tbl\_Persons where id=2

When we delete a record into Tbl\_Persons it will show the value in deleted magic table (Holding). After it will removes from Tbl\_Persons table.

## Update the data in the table

When we update the data in the table with new value, you can see the inserted & and deleted old value.

Create trigger tr\_Tbl\_update on Tbl\_Persons

For update

as

begin

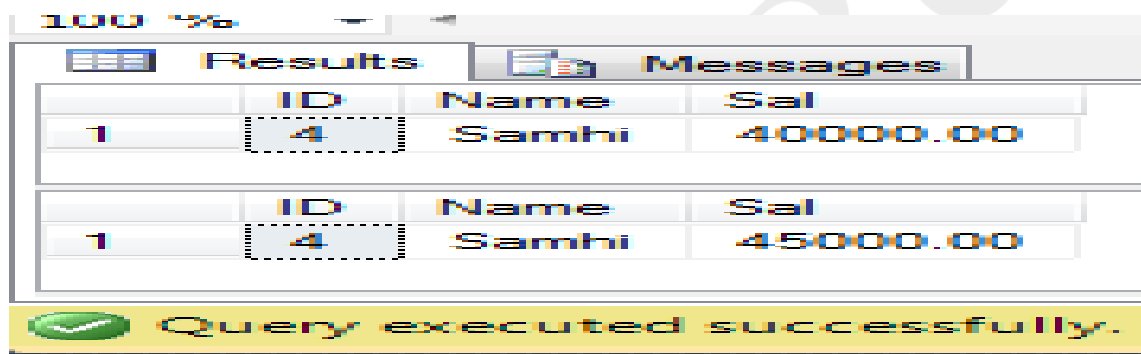
select \* from deleted

select \* from inserted

end

===== Execute the trigger=====

Update Tbl\_Persons set Sal=45000 where id=4;



	ID	Name	Sal
1	4	Samhi	40000.00
1	4	Samhi	45000.00

Query executed successfully.

Here you can clearly notice that how the value is going to update in inserted table & deleted table.

*//Create a trigger which will convert Name in upper case when user insert in any case.*

Create trigger tr\_Tbl\_Persons\_uppercase on Tbl\_Persons

For insert

as

begin

declare @name varchar(50), @sal decimal(10,2)

select @name=name from inserted

update Tbl\_Persons set Name=UPPER(@name)

where Name=@name

end

===== Execute the trigger=====

Insert into Tbl\_Persons values('machavaram',50500);

When user enter any case letter(Upper / Lower case) it will convert it into upper case only with help of Upper() system defined function.

*//Create a trigger which will restrict the insertion operation on a table.*

Create trigger tr\_Tbl\_Persons\_Restrict\_Insert on Tbl\_Persons

After insert

as

begin

print' You cannot insert the values... Insertion Restricted'

rollback Transaction

end

===== Execute the trigger=====

O/P:

You cannot insert the values... Insertion Restricted

Error Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.

*//Create a trigger which will restrict the Updating operation on a table.*

Create trigger tr\_Tbl\_Persons\_Restrict\_Update on Tbl\_Persons

After update

as

begin

print' You cannot perform updation... \*updation Restricted'

rollback Transaction

end

===== Execute the trigger=====

Update Tbl\_Persons set Sal=54000 where id=4

O/P:

(1 row(s) affected)

(1 row(s) affected)

You cannot perform updation... \*updation Restricted

Error Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.

**Note:-**

You can restrict all DML operations on a particular table in a single trigger  
Like below

**Syntax:-**

Create trigger<Trgger\_Name>on<Table\_Name>

After Insert,update,delete

as

begin

print' You Cannot perform DML Operations this Table'

rollback Transaction

end

=====

*//Create a trigger which will restrict the DML operation on a table based on time constraint (if time is after 1pm it's not allowed to update).*

Create trigger tr\_Tbl\_Persons\_DML\_Time on Tbl\_Persons

After insert,update,delete

as

begin

if DATEPART(HH,GETDATE())>13

begin

print' \* Invalid time for perform DML Operations ....'

rollback Transaction

end

end

===== Execute the trigger=====

Insert into Tbl\_Persons values('Syam',24564);

O/P : \* Invalid time for perform DML Operations....

Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.

**Note: -**

- It will not allow performing DML operations on that particular table from 1 PM to 11:59PM (in between 13 to 24 hrs.)
- You can also mention the time between hours like below

If DATEPART(HH,GETDATE()) between 17 and 20

*//Create a trigger which will restrict to updating the salary of person, if the salary is less than old salary.*

```
Create trigger tr_Tbl_Persons_Sal_Update on Tbl_Persons
After update
as
begin
declare @newsal decimal(10,2),@oldsal decimal(10,2)
select @newsal=sal from inserted
select @oldsal=sal from deleted
if(@newsal<@oldsal)
begin
print' New salary is should not be less than old salary '
rollback Transaction
end
end
```

===== Execute the trigger=====

Update Tbl\_Persons set Sal=25000 where id=4;

O/P:

New salary is should not be less than old salary

Error Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.

Note: While executing the above query it will fire a trigger that previously we applied a trigger for Restrict DML operations on this table. SO we should disable that trigger. Then only it will work.



*//Create a trigger which is restricted deletes operation, if the person name is 'Aashrith'.*

Create trigger tr\_Tbl\_Persons\_Delete\_restriction on Tbl\_Persons

After delete

as

begin

declare @name varchar(40)

select @name=name from deleted

if @name='Aashrith'

begin

print ' \* Not allowed to delete this record. ... '

rollback Transaction

end

end

===== Execute the trigger=====

Delete from Tbl\_Persons where Name='Aashrith';

O/P: \* Not allowed to delete this record....

Error Msg 3609, Level 16, State 1, Line 1

The transaction ended in the trigger. The batch has been aborted.

### **Syntax for Disable / Enable / Drop Trigger(s)**

---

#### **Disable:-**

1) Alter Table<Table\_Name>disable trigger ALL;

➤ It will disable all the triggers on that particular table.

E.g:- Alter Table Tbl\_Persons disable trigger ALL;

2) Alter Table<Table\_Name>Disable trigger<Trigger\_Name>;

➤ It will disable mentioned / particular trigger which you want to delete.

E.g:- Alter Table Tbl\_Persons disable trigger

tr\_Tbl\_Persons\_Restrict\_Update;

**Enable:-**

1) Alter Table<Table\_Name>enabletrigger ALL;

- It will enable all the triggers on that particular table.

E.g:-Alter Table Tbl\_Persons enable trigger ALL;

2) Alter Table<Table\_Name>Enable trigger<Trigger\_Name>;

- It will enable mentioned / particular trigger which you want to delete.

E.g:- Alter Table Tbl\_Persons enable trigger  
tr\_Tbl\_Persons\_Restrict\_Update;

**Drop:-**

drop trigger <Trigger\_Name>;

E.g:- drop trigger tr\_Tbl\_Persons\_Delete\_restriction ;

**Alter DML Trigger:-**

Alter keyword is used for to modify the existing trigger body/ context /Functionality.

```
Alter Trigger <Existing_trig_name>  
ON { table_name | view_name }  
{ FOR | AFTER | INSTEAD OF }  
{ [ INSERT ] , [ UPDATE ] , [ DELETE ] }  
AS { sql_statement [ ; ] }  
GO
```

**Renaming Trigger:-**

Syntax:-

sp\_rename 'Old\_Trigger\_Name', 'New\_Trigger\_Name';

E.g:-

sp\_rename'tr\_Tbl\_Persons\_DML\_Time','tr\_Tbl\_Persons\_DML\_Time\_1';

=====

**Instead of triggers:** fires instead of the triggering action. The Insert, Update, and Delete statements, causes an instead of trigger to fire Instead of the respective statement execution.

**Let's create 2 tables for this:-**

Create table tbl\_Employee\_Demo

```
(  
Emp_ID int identity,  
Emp_Name varchar(55),  
Emp_Sal decimal (10,2)  
)
```

Create table Tbl\_Employee\_Demo\_Audit

```
(  
Emp_ID int,  
Emp_Name varchar(55),  
Emp_Sal decimal(10,2),  
Audit_Action varchar(100),  
Audit_Timestamp datetime  
)
```

*//Create a instead of trigger to insert values*

Create trigger tr\_InsteadOf\_Insert ON Tbl\_Employee\_Demo  
instead of insert

as

```
declare @emp_id int, @emp_name varchar(55), @emp_sal decimal(10,2),  
@audit_action varchar(100);
```

```
select @emp_id=i.Emp_ID from insertedi;
```

```
select @emp_name=i.Emp_Name from insertedi;
```

```
select @emp_sal=i.Emp_Sal from insertedi;
```

```
set @audit_action='Inserted Record -- Instead Of Insert Trigger.'
```

```
begin
```

```
begin tran
```

```
if(@emp_sal>=1000)
```

```
begin
```

```
raiserror('Cannot Insert where salary > 1000',16,1);
```

```
rollback;
```

```
end
```

```
elsebegin
```

```
Insert into Tbl_Employee_Demo(Emp_Name,Emp_Sal)values
```

```
(@emp_name,@emp_sal);  
Insert intoTbl_Employee_Demo_Audit(Emp_ID,Emp_Name, Emp_Sal,  
Audit_Action, Audit_Timestamp) values(@ @identity,@emp_name,  
@emp_sal, @audit_action, getdate());  
commit;  
print'Record Inserted -- Instead Of Insert Trigger.'  
end  
end  
===== Execute the trigger=====
```

Insert into Tbl\_Employee\_Demo values ('Aashrith',1300)

When user supplies the values like this it will throw error like below. It will raise error since we are checking salary >=1000

Msg 50000, Level 16, State 1, Procedure tr\_InsteadOf\_Insert, Line 13  
Cannot Insert where salary > 1000  
Msg 3609, Level 16, State 1, Line 1  
The transaction ended in the trigger. The batch has been aborted.

Insert into Tbl\_Employee\_Demo values ('Aashrith',900)

O/P:

(1 row(s) affected)

(1 row(s) affected)

Record Inserted -- Instead Of Insert Trigger.

(1 row(s) affected)

Note:-

Trigger has inserted the new record to Employee\_Demo\_Audit table for insert statement. In this way we can apply business validation on the data to be inserted using Instead of trigger and can also trace an insert activity on a table.

*//Create a instead of trigger to update*

```
Create trigger tr_InsteadOfUpdate ON dbo.Tbl_Employee_Demo
instead of Update
AS
Declare @emp_id int, @emp_name varchar(55), @emp_sal decimal(10,2),
@audit_action varchar(100);
Select @emp_id=i.Emp_ID from insertedi;
Select @emp_name=i.Emp_Name from insertedi;
Select @emp_sal=i.Emp_Sal from insertedi;
Set @audit_action='Updated Record -- Instead Of Updated Trigger.'
begin
begin tran
if(@emp_sal>=2000)
begin
raiserror('cannot update where salary < 1000',16,1);
rollback;
end
else begin
insert into Tbl_Employee_Demo_Audit(Emp_ID,Emp_Name,    Emp_Sal,
Audit_Action,Audit_Timestamp)
values (@emp_id, @emp_name, @emp_sal, @audit_action ,getdate());
commit;

print'record Updated -- Instead Of Update Trigger.'
end
end
===== Execute the trigger=====
update Tbl_Employee_Demo set Emp_Sal=1000 where emp_id=13;

It will throw an error ...if user supply Emp_sal 1000 are above
Msg 50000, Level 16, State 1, Procedure tr_InsteadOfUpdate, Line 12
Cannot Insert where salary > 1000
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

Update Tbl\_Employee\_Demo set Emp\_Sal=950 where emp\_id=13;  
O/P:

(1 row(s) affected)

Record Updated -- Instead Of Update Trigger.

(1 row(s) affected)

*//Create a instead of delete trigger*

Note: - Before creating this trigger you should disable insert trigger ..

Create trigger tr\_InsteadOf\_Delete ON Tbl\_Employee\_Demo  
Instead of delete

AS

Declare @emp\_id int,@emp\_name varchar(55), @emp\_sal  
decimal(10,2),@audit\_action varchar(100);

select @emp\_id=d.Emp\_ID from deletedd;

select @emp\_name=d.Emp\_Name from deletedd;

select @emp\_sal=d.Emp\_Sal from deletedd;

set @audit\_action='Record Deleted-- Instead Of Deleted Trigger.'

Begin tran

if(@emp\_sal>1000)

begin

raiserror('Cannot delete where salary > 1200',16,1);

rollback;

end

else begin

delete from Tbl\_Employee\_Demo where Emp\_ID=@emp\_id;

commit;

insert into Tbl\_Employee\_Demo\_Audit(Emp\_ID,Emp\_Name,  
Emp\_Sal,Audit\_Action, Audit\_Timestamp) values(@emp\_id,  
@emp\_name,@emp\_sal, 'Deleted -- Instead Of Delete  
Trigger.',getdate());

print 'record Deleted -- Instead Of Deleted Trigger.'

end

===== Execute the trigger=====

Delete from Tbl\_Employee\_Demo where emp\_id=14  
When user try deleted the above record it will through an error,  
because

Delete from Tbl\_Employee\_Demo where emp\_id=15  
O/P:

(1 row(s) affected)

(1 row(s) affected)

Record Deleted -- Instead Of Deleted Trigger.

Trigger has inserted the new record to Tbl\_Employee\_Demo\_Audit table for insert statement. In this way we can apply business validation on the data to be inserted using Instead of trigger and can also trace an insert activity on a table.

### ***//Create a instead of trigger to insert values on a View***

Create table Tbl\_emp\_1(id int primary key,Name varchar(40),Gender char,Dept\_id int)

Create table tbl\_dept\_1(Dept\_id int references Tbl\_emp\_1,Dept\_Name varchar(10))

Insert into Tbl\_emp\_1 values(1,'Aashrith','M',2)

Insert into Tbl\_emp\_1 values(2,'Vivaan','M',2)

Insert into Tbl\_emp\_1 values(3,'Samhitha','M',1)

Insert into Tbl\_emp\_1 values(4,'Agasthay','M',3)

Insert into tbl\_dept\_1 values(1,'HR')

Insert into tbl\_dept\_1 values(2,'IT')

Insert into tbl\_dept\_1 values(3,'DBA')

Select \* from tbl\_Emp\_1;

Select \* from tbl\_dept\_1;

## **Creating a trigger on View:-**

```
Create view vw_tbl_Emp_Dept_1 as select id,Name,Gender,Dept_Name  
from tbl_Emp_1,tbl_dept_1
```

```
Where tbl_Emp_1.Dept_id=tbl_dept_1.Dept_id;
```

```
--drop view vw_tbl_Emp_Dept_1
```

```
Select * from vw_tbl_Emp_Dept_1
```

```
Create triggertr_vw_tbl_Emp_Dept_1_instead_of_insert
```

```
On vw_tbl_Emp_Dept_1
```

```
Instead of insert
```

```
as
```

```
begin
```

```
select * frominserted;
```

```
select * fromdeleted;
```

```
end
```

```
===== Execute the trigger=====
```

```
Insert into vw_tbl_Emp_Dept_1 values (6,'Mahathi','F','HR')
```

Results		Messages		
id	Name	Gender	Dept_Name	
9	Dvipayaan	M	IT	

id	Name	Gender	Dept_Name
----	------	--------	-----------

In this process when user try to insert the record into view it will insert into `__inserted` magic table. Now we are altering our trigger.

**Note:** - Generally Complex views are not possible to perform DML operations because that view depends on multiple base tables.

By using triggers only we can perform the DML operations on Complex views like bellow.



**Alter View**

```
Alter trigger tr_vw_tbl_Emp_Dept_1_instead_of_insert
On vw_tbl_Emp_Dept_1
instead of insert
as
begin
declare @dept_id int
select @dept_id=dept_id from Tbl_Dept_1
join inserted on inserted.dept_name=tbl_dept_1.dept_name
if(@dept_idisnull)
begin
raiserror('Invalid Dept Name',16,1);
--if the user supply wrong dept_name it will through error and terminate
from the program.
return
end
inser tinto Tbl_Emp_1(id,Name,Gender,Dept_id)
select id,Name,Gender,@dept_id from inserted
end
===== Execute the trigger=====
```

Insert into vw\_tbl\_Emp\_Dept\_1 values (9,'Dvipayaan','M','IT')

O/p:

(1 row(s) affected)

(1 row(s) affected)

When user inserted a record it will inserted that record into view as well as Tbl\_Emp\_1 also.

**DDL Triggers:-**

- DDL triggers are used to restrict the DDL Operations.
- DDL triggers are fire in response to DDL events like Create, Alter & Drop (Table, Function, Index, Stored Procedure, etc...)
- DDL Triggers can be set with either a Server scope or database scope.
- We can use only FOR/AFTER clause in DDL triggers not INSTEAD OF clause means we can make only After Trigger on DDL statements.
- DDL trigger can be used to observe and control actions performed on the server, and to audit these operations. DDL triggers can be used to manage administrator tasks such as auditing and regulating database operations.

Note: - If you want check all the list DDL events visit the bellow link.  
<https://msdn.microsoft.com/en-us/library/bb522542.aspx>

**Syntax:-**

```
Create Trigger trig_name  
ON {Scope (SERVER | DATABASE)}  
{ For (Event(s)...) }  
As begin{ sql_statement ; }  
End
```

Eg:- Events are like Create\_Table, Alter\_Table, Drop\_Table

**Uses of DDL Triggers:-**

- If you want to execute some code in response to a specific DDL event.
- To prevent certain changes to your database schema.
- Audit the changes that the users are making to the database structure.

**Examples on DDL Triggers on a Database scoped...**

*//Create a DDL trigger whenever user creates a table it will show a message.*

```
Create trigger tr_create_table on database  
For Create_Table  
As begin  
print'New Table created in the Database'  
end
```

===== Execute the trigger=====

```
Create table Tbl_Demo(id int,name varchar(20));
```

O/P: New Table created in the Database

*//Create a DDL trigger whenever user creates modifies or deletes a table it will show a message.*

Alter trigger tr\_create\_table on database

For Create\_Table, alter\_table, drop\_table

As begin

print'The Table created, modified or deleted in the DB ';

end

===== Execute the trigger=====

Drop table Tbl\_Demo;

O/P: The Table created, modified or deleted in the DB

*//Create a DDL trigger to restrict the user to create / alter / drop the tables options .*

Create trigger tr\_Restrict\_table\_Options on database

For Create\_Table,alter\_table,drop\_table

As begin

rollback

print'\*\*You cannot create, alter or drop the a Table.. '

end

===== Execute the trigger=====

Create table Tbl\_Demo(id int,name varchar(20));

O/p :               \*\*You cannot create, alter or drop the a Table..

Msg 3609, Level 16, State 2, Line 1

The transaction ended in the trigger. The batch has been aborted.

Note:-

Certain system stored procedures that perform DDL -like operations can also fires DDL triggers.

*//Create a DDL trigger to rename options ...*

Create trigger tr\_Rename\_Options on database

For rename

As begin

```
print '**You just renamed something.. '
```

```
end
```

```
===== Execute the trigger=====
```

```
1) Create table Tbl_Demo(id int,name varchar(20))
```

```
2) sp_Rename'Tbl_Demo_New.id','No','Column'
```

O/P:

Caution: Changing any part of an object name could break scripts and stored procedures.

**\*\*You just renamed something..**

### **Examples on DDL Triggers on Server scoped...**

Creating server scope trigger is similar to database scope trigger. For that user should change the scope from Database to all server.

*//Create a DDL trigger to restrict the user to create / alter / drop the table's options on server scope.*

Create trigger tr\_Server\_Scope\_Restrict on all server

For Create\_Table,alter\_table,drop\_table

As begin

rollback

```
print '**You cannot create, alter or drop the a Table.. '
```

```
end
```

```
===== Execute the trigger=====
```

```
Create table Tbl_Demo(id int,name varchar(20))
```

O/P:

**\*\*You cannot create, alter or drop the a Table..**

Msg 3609, Level 16, State 2, Line 1

The transaction ended in the trigger. The batch has been aborted.

Note:-

User cannot create, alter or drop any table in any database.

**Disable / Enable / Drop triggers:-**

Disable Trigger<Trigger\_Name>ON database // particular trigger

Disable Trigger ALL ON database // All trigger's

Enable Trigger<Trigger\_Name>ON database // particular trigger

Enable Trigger ALL ON database // All trigger's

Drop trigger<Trigger\_Name>on database

**Server Scope:-**

Disable Trigger<Trigger\_Name>ON database // particular trigger

Disable TriggerALL ON database

Enable TriggerALL ON database

Drop Trigger<Trigger\_Name>ON all server

**Logon Triggers**

Logon triggers are special type of trigger that fire when LOGON event of SQL Server is raised. This event is raised when a user session is being established with SQL Server that is made after the authentication phase finishes, but before the user session is actually established. Hence, all messages that we define in the trigger such as error messages, will be redirected to the SQL Server error log.

Logon triggers do not fire if authentication fails. We can use these triggers to audit and control server sessions, such as to track login activity or limit the number of sessions for a specific login.

Select is\_user\_Process,original\_login\_name , \* from  
sys.dm\_exec\_sessions order by login\_time desc;

This is predefined view to get sessions list..

*//Create a Logon trigger to limit the maximum number of open connections for a user to 3*

Create trigger tr\_con\_login\_audit on all server

For logon

As begin

Declare @Loginname nvarchar(100)

Set @Loginname=original\_login()

if(

select count(\*) from sys.dm\_exec\_sessions where is\_user\_Process=1  
and original\_login\_name=@Loginname)>3

begin

print'Fourth connection attempt by '+@Loginname+' blocked'

rollback

end

end

===== Execute the trigger=====

With this process user cannot open the sessions more than 3

Drop trigger<Trigger-Name>on all server

**Note:-**

The trigger error message will be written to the error log, to read the error log execute the predefined system stored procedure and find the error msg in that list.

Execute sp\_readerrorlog

**SQL Server Setting Triggers Firing Order**

Sometimes we have multiple triggers on the same event(s). In that case we can't predict the firing order of triggers. Sometimes the firing order of trigger(s) is important for us to implement business logic. To solve this issue, In SQL Server we have option to set the firing order of triggers on same event(s).

- Server scoped triggers will always fire before any of the database scoped triggers.
- Using the `sp_settriggerorder` stored procedure, you can set the execution order of server scoped / database scoped triggers.

**Syntax:-**

```
sp_settriggerorder @triggername='trg_name',  
@order='FIRST|LAST|NONE',  
@stmttype='INSERT|UPDATE|DELETE|CREATE_INDEX,  
ALTER_INDEX', @namespace='DATABASE|SERVER|NULL';
```

**Example:-**

```
Create table Tbl_MyTest  
(  
ID int NOT NULL,  
Description varchar(100)  
)
```

--Now create triggers on above created table at same event INSERT

```
create trigger tr_TriggerOrder_1  
on Tbl_MyTest  
after insert  
as  
print 'i will be fired first.'  
go  
create trigger tr_triggerorder_2  
on tbl_mytest  
after insert  
as  
print 'i will be fired last.'  
go
```

```
create triggertr_triggerorder_3
on tbl_mytest
after insert
as
print'i won't be first or last.'
go
===== execute the trigger=====
--now set triggers firing orders
```

```
Exec sp_settriggerorder 'tr_TriggerOrder_1','First','INSERT'
Exec sp_settriggerorder 'tr_TriggerOrder_2','Last','INSERT'
```

```
--The order of firing the third trigger 'dbo.trg_i_TriggerOrder3' will be
between above two
--Insert data to see trigger firing order
```

```
Insert Tbl_MyTest(ID , Description)values (1,'Trigger firing order')
```

O/P:

```
I will be fired first.
I won't be first or last.
I will be fired last.
```

```
(1 row(s) affected)
```



**Note:-**

If you have a database scoped and a server based trigger handling the same event, and if you have set the execution order at both the levels. Here is the execution order of the triggers

- The server scoped trigger marked first
- Other server scope triggers
- The server scope trigger marked last
- The database scope trigger marked first
- Other database scope triggers
- The database scope trigger marked last.

**Uses of Triggers:-**

- 1) Using triggers we can enforce business rules that can't be defined by using integrity constants.
- 2) Using triggers we can gain strong control over the security.
- 3) We can also collect statistical information on the table access.
- 4) We can automatically generate values for derived columns such as auto increment numeric primary key.
- 5) Using triggers you can prevent invalid transaction.

**Restriction on Triggers**

- Maximum size of the trigger body must not exceed 32,760 bytes because triggers' bodies are stored in LONG datatypes columns.
- A trigger may not issue transaction control statements or TCL statements such as COMMIT, ROLLBACK or SAVEPOINT. All operations performed when the trigger fires, become part of a transaction. Therefore whenever this transaction is rolled back or committed it leads to the respective rolling back or committing of the operations performed.

## **Exception Handling**

- Exception is a run time error
- Generally in any programming language as well as database also 2 types of errors will occur
  - Compilation error
  - Runtime error

Compilation error:-

- Compile time error will occur at the time of compilation of program.
- Programmer can easily solve the compilation errors by reading the error message.

Runtime error:-

- These errors will occur at the time of execution of the program.
- We can't rectify the runtime errors but we can handle the runtime.

Note:-

SQL Server 2005 introduced TRY...CATCH statement which helps us to handle the errors effectively in the back end. This handling of the exception can provide additional information about the errors.

### **TRY...CATCH**

The TRY...CATCH statement works the same as in the programming languages. First it gets executed in the SQL statement which we have written in the TRY block and if any error occurs, then it will get executed the CATCH block.

Syntax:-

```
BEGIN TRY
--//SQLStatements
END TRY
BEGIN CATCH
--//Handletheexceptiondetails
END CATCH
```

Normally, SQL Server stores the default error messages which occurred in the execution in the following system table:

Select \* from sys.messages;

But we can create our own error message details with the help of this exception handling.

*// create a procedure for dividing the given two values by using Try, Catch implementation with user defined error statemnts.*

```
Create proc spDiv_User_er_msg
@a int,@b int
as
begin
declare @cint
begin try
set@C=@a/@b
print'Division values is:'+cast(@c as varchar)
end try
begin catch
Print'Second Number should not be Zero'
End catch
end
```

===== Execute the Procedure =====

```
Exec spDiv_User_er_msg20,10
o/p: Division values is:2
```

```
exec spDiv_User_er_msg 20,0
o/p: Second Number should not be Zero
```

*// create a procedure for dividing the given two values by using Try, Catch implementation with system defined error statements.*

```
Create proc spDiv_Sys_er_msg
@a int,@b int
as
begin
declare @cint
```

```
begin try
set @C=@a/@b
print'Division values is:'+cast(@c as varchar)
end try
begin catch
Print error_message()
End catch
end
```

===== Execute the Procedure =====

Exec spDiv\_Sys\_er\_msg20,0

O/p: Divide by zero error encountered.

### **Guidelines for using TRY ... CATCH Construct:-**

- A Try block must be followed by a Catch block.
- Each Try ...Catch construct must be inside a single batch, stored procedure, or trigger.
- Try... Catch constructs can be nested, Try... Catch constructs can be placed inside other Try and Catch blocks.
- To handle an error that occurs within a given Catch block, write Try...Catch block within the specified Catch block.

### **Error Functions:-**

Try... Catch uses the following error functions to capture error information:

- **Error\_number()**  
Returns the error number →int
- **error\_message()**  
Returns the complete text of the error message→  
nvarchar(4000)
- **error\_severity()**  
Returns the error severity →int

- Error\_State()  
Returns the error state number →int
- Error\_Line()  
Returns the line number inside the routine that caused the error  
→int
- Error\_Procedure()  
Returns the name of the stored procedure or trigger where the error occurred → nvarchar(128)

#### Example 1:-

=====

```
begin
select 1/0
end
```

if user try to execute the above example it will through an system defined error like bellow

Msg 8134, Level 16, State 1, Line 3  
Divide by zero error encountered.

=====

#### Example 2:-

```
begin
begin try
select 1/0
end try
begin catch
select 'Error Handled'
end catch
end
```

when user execute the above program it will return user defined message  
Error handled

Example 3:-

```
Begin try
select 1/0
end try
begin catch
select
ERROR_LINE(),
ERROR_MESSAGE(),
ERROR_NUMBER(),
ERROR_PROCEDURE()as'proc',
ERROR_SEVERITY(),
ERROR_STATE()
End catch
end
```

(No column name)	(No column name)	(No column name)	proc	(No column name)	(No column name)
3	Divide by zero error encountered.	8134	NULL	16	1

Example 3:-

//Creating procedure for the above predefined function to call in any procedure.

```
Create proc spmy_EXPas
begin
begin try
select 1/0
end try
begin catch
select
```

```
ERROR_LINE(),
ERROR_MESSAGE(),
ERROR_NUMBER(),
ERROR_PROCEDURE()as'proc',
ERROR_SEVERITY(),
ERROR_STATE()
endcatch
end
===== Execute Proc=====
```

```
Alter proc spmy_EXPas
begin
begin try
select 1/0
end try
begin catch
exec sperror_handler
end catch
end
```

```
exec spmy_EXP
```

-----

```
Alter proc sperror_handleras
begin

select
ERROR_LINE(),
ERROR_MESSAGE(),
ERROR_NUMBER(),
ERROR_PROCEDURE()as'proc',
ERROR_SEVERITY(),
ERROR_STATE()
end
```

-----

```
Exec sperror_handler
```





Results		Messages				
	(No column name)	(No column name)	(No column name)	proc	(No column name)	(No column name)
1	NULL	NULL	NULL	NULL	NULL	NULL

Working with the **THROW** Statement:-

To simplify returning errors in a CATCH block, SQL Server 2012 introduced the THROW statement. With the THROW statement, you don't have to specify any parameters and the results are more accurate. You simply include the statement as is in the CATCH block.

NOTE: You can use the THROW statement outside of the CATCH block, but you must include parameter values to do so.

### Syntax:-

```
BEGIN TRY
-- write your SQL statements.
END TRY
BEGIN CATCH
THROW;
END CATCH
```

### Example:-

```
create table dbo.testrethrow
(id int primarykey
);
Begin try
Insert dbo.testrethrow(id)values(1);
-- Force error 2627, Violation of PRIMARY KEY constraint to be
raised.
Insert dbo.testrethrow(id)values(1);
endtry
begin catch

print'in catch block.';
throw;
end catch;
```

O/P:

```
(1 row(s) affected)
(0 row(s) affected)
```

In catch block.

Msg 2627, Level 14, State 1, Line 7

Violation of PRIMARY KEY constraint 'PK\_TestReth\_3214EC27EC14DA9B'. Cannot insert duplicate key in object 'dbo.TestRethrow'. The duplicate key value is (1).

### **Raiserror:-**

Raiserror(\_Error Message', ErrorSeverity,ErrorState)

Create and return custom errors

Severitylevel=16` 9indicates general errors that can be corrected by the user)

State=number between 1 & 255 . Raiserror only generates errors with state from 1 through 127.

### **Differences between RAISERROR and THROW in Sql Server**

Both RAISERROR and THROW statements are used to raise an error in Sql Server. The journey of RAISERROR started from Sql Server 7.0, whereas the journey of THROW statement has just began with Sql Server 2012. Obviously, Microsoft suggesting us to start using THROW statement instead of RAISERROR. THROW statement seems to be simple and easy to use than RAISERROR.

THROW	RAISE ERROR
<p>THROW is introduced with SQL Server 2012. It is very simple and easy to use.</p> <p>We can re-throw the original exception that was caught with in the TRY...CATCH block. To do this just specify the THROW without a parameter.</p> <p>Example</p> <pre>BEGIN TRY   DECLARE @result INT   --Generate casting error   SET @result = 'This is test' END TRY BEGIN CATCH   THROW END CATCH</pre> <p>Result</p> <p>Msg 245, Level 16, State 1, Line 16 Conversion failed when converting the varchar value 'This is test' to data type int.</p>	<p>RAISE ERROR was introduced with SQL Server 2005</p> <p>We cannot re-throw the original exception that is invoked the CATCH block. It always raises a new exception and the result, original exception is lost.</p> <p>Example</p> <pre>BEGIN TRY   DECLARE @result INT   --Generate casting error   SET @result = 'This is test' END TRY BEGIN CATCH   DECLARE     @ErrorMessage NVARCHAR(2048),     @ErrorSeverity INT,     @ErrorState INT   SELECT     @ErrorMessage = ERROR_MESSAGE(),     @ErrorSeverity = ERROR_SEVERITY(),     @ErrorState = ERROR_STATE()    RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState) END CATCH</pre> <p>Result</p> <p>Msg 50000, Level 16, State 1, Line 16 Conversion failed when converting the varchar value 'This is test' to data type int.</p>
There is no severity parameter. The exception severity is always set to 16 until re-throwing from a CATCH block	The Severity parameter specified the severity of the exception
It requires a semicolon (;) as a statement terminator. The statement before the THROW must have a semicolon.	It does not require any statement terminator.
We can raise a user defined error message with a new message id without defining it in the sys.messages table.	RAISERROR accepts a message id or string but when we use a message id it must be defined in the sys.Messages table.

**Transferring data from one system to another system**

Whenever there is a need if you want to transfer the data from one system to another system we have some mechanisms to transfer.

**1. By using data files (.mdf & .ldf files)**

In this mechanism we should follow two mechanisms

- Detaching method
- Attaching method

**Steps to „Detach“ the database from the server location:-**

- Open SQL Server Management Studio
- Go to Object Explorer
- Go to database folder
- Select data base folder which you want to move(detach)
- Click Right button
- Go to Task option
- Click on Detach
- Enable \_Drop' & \_Update' check box
- Click \_OK'
- With this process selected database will detach from database folder
- Now go to the file location place  
C:\Program Files\Microsoft SQL  
Server\MSSQL10\_50.SQLEXPRESS\MSSQL\DATA
- From this folder you can select the file(.mdf & .ldf) which you want to move

**Steps to „Attach“ the database from the server location:-**

- Open SQL Server Management Studio
- Go to Object Explorer
- Go to database folder
- Click right button
- Select Attach
- Click \_Add' button
- Select the \_.mdf' which you want to add your current server system
- Click \_OK' 2times
- Refresh the database folder, you should find the attached database in that folder.

## 2. By Backup file(.bak)

In this mechanism user should follow the following two methods

- Creating backup file
- Restoring backup file

### Steps to creating backup file:-

- Open SQL Server Management Studio
- Go to Object Explorer
- Go to database folder
- Select data base folder which you want to take backup
- Click right button
- Go to Tasks' -> select & Click on Back up...'
- Click OK'
- Click OK'
- Now go to the file location place  
C:\Program Files\Microsoft SQL  
Server\MSSQL10\_50.SQLEXPRESS\MSSQL\Backup
- Select the backup file(.bak) which you want to take back up

### Steps to creating backup file:-

- Open SQL Server Management Studio
- Go to Object Explorer
- Go to database folder Click right button
- Select Restore Database'
- Select Device' Radio button
- Click on browse [...] to select database
- Click on Add' button
- Select the folder where that folder contain backup' bak file extension
- Select the bak' file
- Click OK'
- Click OK'
- Check the Enable button weather it enable or not ( By default it will enable)
- Click OK
- Click OK
- Refresh the database folder, you should find the Added database folder in that list.

### 3. Generate Scripts... file(.bak)

In this mechanism user should follow the following two methods

- Generate Script file
- Executing Generated Script file

#### Steps to creating backup file:-

- Open SQL Server Management Studio
  - Go to Object Explorer
  - Go to database folder
  - Select data base folder which you want to take backup
  - Click right button
  - Go to Tasks' -> select & Click on Generate Scripts...'
  - Click Next
  - Select Radio button ☐ Script entire database and all database object'
  - Click Next'
  - At Set Scripting Options' wizard click on Advanced' button
  - In that options set Types of data to script" as : **Schema and data**
  - Click OK'
  - Click on Next'
  - Click on Next'
  - Click on Finish'
  - With this process it will create a script file in C' drive Documents' folder..
- C:\Users\welcome\Documents\script.sql (Extension with .sql)

#### Executing Generated Script file

- Select the Script file (.sql file )
- Double click the file
- It will open in SQL window
- Select in Which data base it is going to Execute
- Click on Execute (F5)

**Primary data file (.mdf file):-**

It will store all database tables data and will save with an extension of .mdf (Master data file)

**Log data file (.ldf file):-**

It will store transaction or overall query information, which was executed by the user in database objects (tables, Views, Synonyms) and will save with an extension of .ldf .

Root location of .MDF and .LDF files:-

C:\Program Files\Microsoft SQL Server\MSSQL10\_50.SQLEXPRESS\MSSQL\DATA

→ **The above two files are used for transforming the database information from one system to another system or else one location to another location.**

***Executing Script Files from command Prompt***

In the command prompt window the SQL script files can be executed using "**sqlcmd**" command.

*Syntax for the sqlcmd command is:-*

```
sqlcmd -S <serverName> -U <UserName> -P <Password>  
-i <inputqueryfilename> -o <outputfilename>
```

sqlcmd --> command to run sql file from command prompt

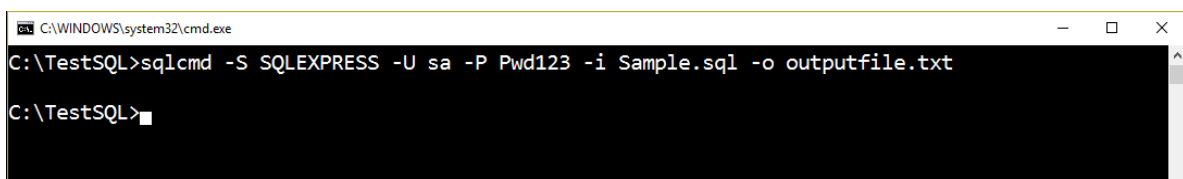
-S ---> server name

-U ---> User name

-P ---> password

-i ---> input sql file name

-o ---> name of the output file to be created



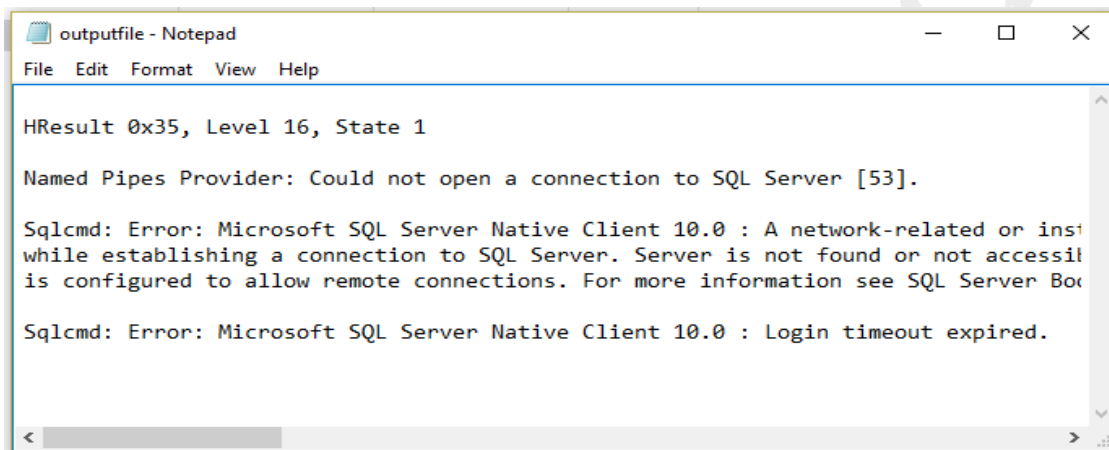
```
C:\WINDOWS\system32\cmd.exe  
C:\TestSQL>sqlcmd -S SQLEXPRESS -U sa -P Pwd123 -i Sample.sql -o outputfile.txt  
C:\TestSQL>
```

The sql file will be executed and the output will be stored in the output file created with the specified name.

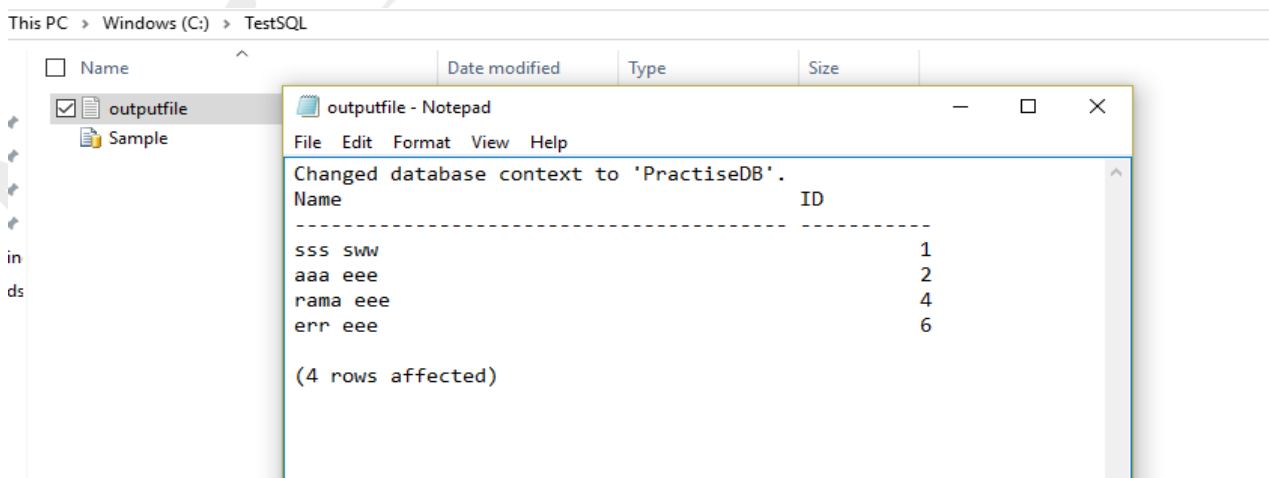
For example: Sample.sql file

```
Sample.sql - SAMHI...actiseDB (sa (53))
Use PractiseDB;
go
select (fname + ' ' + Lname) as Name, ID from Student
where age = 20 or age = 5
```

In case of error the output file will be created with the error message



If there are no errors and the sql file gets executed successfully then the output file will be created with the result.







# Interview Question & Answers

Mr. R. J. Reddy

### 1. What is DBMS?

A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

### 2. What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

Example: SQL Server.

### 3. What is SQL?

SQL stands for Structured Query Language, and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.

### 4. What is a Database?

Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways.

Example: School Management Database, Bank Management Database.

### 5. What are tables and Fields?

A table is a set of data that are organized in a model with Columns and Rows. Columns can be categorized as vertical, and Rows are horizontal. A table has specified number of column called fields but can have any number of rows which is called record.

Example:

Table: Employee.

Field: Emp ID, Emp Name, Date of Birth.

Data: 201456, Aashrith, 06/10/1986.

### 6. What is a primary key?

A primary key is a combination of fields which uniquely specify a row. This is a special kind of unique key, and it has implicit NOT NULL constraint. It means, Primary key values cannot be NULL.

---

### 7. What is a unique key?

A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns.

A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key.

There can be many unique constraint defined per table, but only one Primary key constraint defined per table.

### 8. What is a foreign key?

A foreign key is one table which can be related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table.

### 9. What is a join?

This is a keyword used to query data from one or more tables based on the relationship between the fields of the tables. Keys play a major role when JOINS are used.

### 10. What are the types of join and explain each?

There are various types of join which can be used to retrieve data and it depends on the relationship between tables.

**Inner join:** Inner join return rows when there is at least one match of rows between the tables.

**Right Join:** Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it return s all the rows from the right hand side table even though there are no matches in the left hand side table.

**Left Join:** Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

**Full Join:** Full join return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

### 11. What is normalization?

Normalization is the process of minimizing redundancy and dependency by organizing fields and table of a database. The main aim of Normalization is to add, delete or modify field that can be made in a single table.

---

### 12. What is De-normalization?

De-Normalization is a technique used to access the data from higher to lower normal forms of database. It is also process of introducing redundancy into a table by incorporating data from the related tables.

### 13. What are all the different normalizations?

The normal forms can be divided into mainly 3 forms, and they are explained below

#### **First Normal Form (1NF):**

This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

#### **Second Normal Form (2NF):**

Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

#### **Third Normal Form (3NF):**

This should meet all requirements of 2NF. It contains only columns that are non-transitively dependent on the primary key.

Transitive dependence means a column's value relies upon another column through a second intermediate column.

#### **Example:**

Consider three columns: AuthorNationality, Author, and Book. Column values for AuthorNationality and Author rely on the Book; once the book is known, you can find out the Author or AuthorNationality. But also notice that the AuthorNationality relies upon Author. That is, once you know the Author, you can determine their nationality. In this sense then, the AuthorNationality relies upon Book, via Author. This is a transitive dependence.

### 14. What is a View?

A view is a virtual table which consists of a subset of data contained in a table. Views are not virtually present, and it takes less space to store. View can have data of one or more tables combined, and it is depending on the relationship.

---

### 15. What is an Index?

An index is performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and it will be faster to retrieve data.

Simply put, an index is a pointer to data in a table. 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.

### 16. What are all the different types of indexes?

There are three types of indexes -.

#### **Unique Index.**

This indexing does not allow the field to have duplicate values if the column is unique indexed. Unique index can be applied automatically when primary key is defined.

#### **Clustered Index.**

This type of index reorders the physical order of the table and search based on the key values. Each table can have only one clustered index.

The index can contain multiple columns (a composite index). Whenever we apply Primary key constraint on a column in a table, then automatically clustered index will be applied on primary key column and clustered index will arrange the data in Ascending order.

#### **Non-Clustered Index.**

Non-Clustered Index does not alter the physical order of the table and maintains logical order of data. Each table can have 999 nonclustered indexes.

### 17. What is a Cursor?

Cursor is memory location for storing data bases tables. Cursors is temporary work area allotted to the client at server when a select statement is executed. A cursor contains information of select statement and information of data accessed by it. A cursor can hold more than one row but can process only one row at a time. The set of rows the cursor holds is called as result set.

---

## Types of cursor

1. Implicit Cursor
2. Explicit Cursor

**Implicit sCursor** - The cursors which will be created by SQL by default when select statement executes.

**Explicit Cursor** - The cursor which is created by the user is called as Explicit Cursor. By using this we can fetch multiple rows from the table & store in SQL memory area. Whenever we want to go for 'record by record manipulation' we will use this.

### 18. What is a relationship and what are they?

Database Relationship is defined as the connection between the tables in a database. There are various data basing relationships, and they are as follows:

- One to One
- One to Many
- Many to One
- Self-Referencing

### 19. What is a query?

A DB query is a code written in order to get the information back from the database. Query can be designed in such a way that it matched with our expectation of the result set.

### 20. What is subquery?

A subquery is a query within another query. The outer query is called as main query, and inner query is called subquery. SubQuery is always executed first, and the result of subquery is passed on to the main query.

### 21. What are the types of subquery?

There are two types of subquery – Correlated and Non-Correlated.

A correlated subquery cannot be considered as independent query, but it can refer the column in a table listed in the FROM the list of the main query.

A Non-Correlated sub query can be considered as independent query and the output of subquery are substituted in the main query.

Correlated subquery will work on (n-1) mechanism whereas non correlated subqueries are working on n+1 mechanism.

---

## 22. What is a stored procedure?

Stored Procedure is a function consists of many SQL statement to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required.

## 23. What is a trigger?

Triggers are database object. Basically these are special type of stored procedure that are automatically fired/executed when a DDL or DML command statement related with the trigger is executed. Triggers are used to assess/evaluate data before or after data modification using DDL and DML statements. These are also used to preserve data integrity, to control server operations, to audit a server and to implement business logic or business rule.

## 24. What is the difference between DELETE and TRUNCATE commands?

DELETE	TRUNCATE
DELETE is a DML command.	TRUNCATE is a DDL command.
We can use WHERE clause in DELETE command.	We cannot use WHERE clause with TRUNCATE
DELETE statement is used to delete row(s) from a table	TRUNCATE statement is used to remove all the rows from a table.
DELETE is slower than TRUNCATE statement.	TRUNCATE statement is faster than DELETE statement.
You can rollback data after using DELETE statement.	It is not possible to rollback after using TRUNCATE statement.

## 25. What are local and global variables and their differences?

Local variables are the variables which can be used or exist inside the function. They are not known to the other functions and those variables cannot be referred or used. Variables can be created whenever that function is called.

Global variables are the variables which can be used or exist throughout the program. Same variable declared in global cannot be used in functions. Global variables cannot be created whenever that function is called.

---

## 26. What is a constraint?

Constraints are used to restrict the insertion of unwanted data in any columns. We can create constraints on single or multiple columns of any table. Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table. It maintains the data integrity of the table. This ensures the accuracy and reliability of the data in the database.

### Types of Constraints

- 1) Not Null Constraint  
Indicates that a column cannot store NULL value
- 2) Default Constraint  
Provides a default value for a column when none is specified.
- 3) Unique Constraint  
Ensures that all values in a column are different.
- 4) Primary key Constraint  
A combination of a NOT NULL and UNIQUE. i.e. Uniquely identified each row/record in a database table.
- 5) Foreign key Constraint  
Ensure the referential integrity of the data in one table to match values in another table.
- 6) Check Constraint  
The CHECK constraint ensures that all values in a column satisfy certain conditions.

## 27. What is data Integrity?

Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

## 28. What is Auto Increment?

Auto increment keyword allows the user to create a unique number to be generated when a new record is inserted into the table. AUTO INCREMENT keyword can be used in Oracle and IDENTITY keyword can be used in SQL SERVER.

Mostly this keyword can be used whenever PRIMARY KEY is used.

---



For a given identity property with specific seed/increment, the identity values are not reused by the engine. If a particular insert statement fails or if the insert statement is rolled back then the consumed identity values are lost and will not be generated again. This can result in gaps when the subsequent identity values are generated.

### 29. What is the difference between Cluster and Non-Cluster Index?

Clustered index	Non-Clustered Index
One table can have only one clustered index.	You can have more than one Non-clustered index.
Clustered index is faster.	Non-clustered index has to refer back to the table, if the selected column is not present in the index so its slower.
Clustered index determines the storage order of rows in the table, and hence doesn't require additional disk space.	Nonclustered index is stored separately from the table, additional storage space is required.

### 30. What is Self-Join?

The SELF JOIN is used to join a table to itself as if the table were two tables, using aliases for at least one table in the SQL statement. This is also useful for comparisons within a table.

### 31. What is Cross-Join?

Cross join defines as Cartesian product where number of rows in the first table multiplied by number of rows in the second table. If suppose, WHERE clause is used in cross join then the query will work like an INNER JOIN.

### 32. What is user defined functions?

User Defined functions can be used to perform a complex logic, can accept parameters and return data. Many a times we have to write complex logic which cannot be written using a single query. In such scenarios, UDFs play an important role. For example, we can call user defined function in a where clause or use a user defined function in a JOIN [Where UDF returns a result set].

### 33. What are all types of user defined functions?

Three types of user defined functions are.

Scalar Functions.

Inline Table valued functions.

Multi statement valued functions.

Scalar Functions: - The function which returns a Scalar/Single value.

Table Valued Functions: - This function will take the input from the user and return multiple rows from the table. The return type of table valued function is Table. In this function we are not having `_as begin'` and `_end'` block.

Table Valued Functions can be written as

- Inline Table
- Multi-statement Table

### 34. What is collation?

Collation is defined as set of rules that determine how character data can be sorted and compared. This can be used to compare A and, other language characters and also depends on the width of the characters.

ASCII value can be used to compare these character data.

### 35. What are all different types of collation sensitivity?

Following are different types of collation sensitivity -.

Case Sensitivity – A and a and B and b.

Accent Sensitivity.

Kana Sensitivity – Japanese Kana characters.

Width Sensitivity – Single byte character and double byte character.

### 36. Advantages and Disadvantages of Stored Procedure?

A stored procedure is a data base object which contains precompiled SQL queries. Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of

---

executing multiple queries. This reduces network traffic and provides better security to the data.

Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

### 37. What is Online Transaction Processing (OLTP)?

Online Transaction Processing or OLTP manages transaction based applications which can be used for data entry and easy retrieval processing of data. This processing makes like easier on simplicity and efficiency. It is faster, more accurate results and expenses with respect to OTLP.

Example – Bank Transactions on a daily basis.

### 38. What is CLAUSE?

SQL clause is defined to limit the result set by providing condition to the query. This usually filters some rows from the whole set of records.

Example – Query that has WHERE condition

Query that has HAVING condition.

### 39. What is recursive stored procedure?

A stored procedure which calls by itself until it reaches some boundary condition. This recursive function or procedure helps programmers to use the same set of code any number of times.

### 40. What is Union, minus and Interact commands?

UNION operator is used to combine the results of two tables, and it eliminates duplicate rows from the tables. In case of union, number of columns and data type must be same in both the tables.

The SQL MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement. Each SELECT statement will define a dataset. The MINUS operator will retrieve all records from the first dataset and then remove from the results all records from the second dataset.

---

Intersect operation is used to combine two Select statements, but it only returns the records which are common from both Select statements. In case of Intersect the number of columns and datatype must be same.

#### 41. What is an ALIAS command?

ALIAS name can be given to a table or column. This alias name can be referred in WHERE clause to identify the table or column.

Example-.

```
Select st.StudentID, Ex.Result from student st, Exam as Ex where st.studentID = Ex. StudentID
```

Here, st refers to alias name for student table and Ex refers to alias name for exam table.

#### 42. What's the difference between TRUNCATE and DROP statements?

TRUNCATE removes all the rows from the table, but the structure of the table remains and it cannot be rolled back.

DROP command removes a table from the database and operation cannot be rolled back.

#### 43. What are aggregate and scalar functions?

Aggregate functions are used to evaluate mathematical calculation and return single values. This can be calculated from the columns in a table. Scalar functions return a single value based on the input value.

Example -.

Aggregate – max(), count – Calculated with respect to numeric.

Scalar – UCASE(), NOW() – Calculated with respect to strings.

#### 44. How can you create an empty table from an existing table?

Example will be -.

```
Select * into <New_Table> from <Old_table_Name> where 1=2
```

---

Here, we are copying old table to another table with the same structure with no rows copied.

#### 45. How to fetch common records from two tables?

Common records result set can be achieved by -.

```
Select studentID from student. <strong>INTERSECT </strong>
Select StudentID from Exam
```

#### 46. How to fetch alternate records from a table?

Records can be fetched for both Odd and Even row numbers -.

To display even numbers-.

```
SELECT empid,EmpNAME,ROW FROM(
SELECT ROW_NUMBER()OVER (ORDER BY empid)AS ROW,* FROM
Employee) A WHERE ROW%2=0
```

To display odd numbers-.

```
SELECT empid,EmpNAME,ROW FROM(
SELECT ROW_NUMBER()OVER (ORDER BY empid)AS ROW,* FROM
Employee) A WHERE ROW%2=1
```

#### 47. How to select unique records from a table?

Select unique records from a table by using DISTINCT keyword.  
Select DISTINCT StudentID, StudentName from Student.

#### 48. What is the command used to fetch first 5 characters of the string?

There are many ways to fetch first 5 characters of the string -.  
Select SUBSTRING(StudentName,1,5) as studentname from student  
Select RIGHT(Studentname,5) as studentname from student

#### 49. Which operator is used in query for pattern matching?

LIKE operator is used for pattern matching, and it can be used as -.  
% – Matches zero or more characters.  
\_(Underscore) – Matching exactly one character.

Example -.

```
Select * from Student where studentname like _a%`  
Select * from Student where studentname like _ash_`
```

#### 50. Where can we use Inline Table Valued functions?

Inline Table valued functions can be used to achieve the functionality of parameterized views.

The table returned by the table valued function, can also be used in joins with other tables.

#### 51. Can we create a foreign key without Primary key?

Yes, if the table has unique key then it is possible to create a foreign key constraint.

#### 52. What is check constraint?

Check constraint specifies a condition that is enforced for each row of the table on which the constraint is defined. Once constraint is defined insert, update to the data within the table is checked against the defined constraint.

#### 53. is it possible to create Cluster Index on Unique key column?

Yes, unique key column by default creates a non-cluster index on the column. Primary key creates cluster index on the column. The only difference is Unique key column allows only one null value, though it still maintains the uniqueness of the column.

#### 54. How to get a row was inserted most recently in a table?

```
Select top 1 * from <Table_Name> order by <Column_Name>  
desc;
```

#### 54. How to get nth record in a table?

First get the n records from the table using

```
Select Top n from <table_name>
```

Now reverse the order using identity column like:

```
Select Top n from (table_name> order by 1 desc;
```

Now we need nth record that can be get as

```
Select top 1 * from (Select Top n * from <table_Name> order by 1  
desc) <Alias Name>
```

55. What is the page size in SQL Server?

8 Kb

56. Which function is used to count more than two billion rows in a table?

Count\_big()

57. Can we use truncate command on a table which is referenced by foreign key?

No, we cannot use truncate command on a table with foreign key because of referential integrity.

58. Can we use 'Print' statement in Function?

No.

59. select \* from Tbl\_Emp where Cell\_No like '5\_\_\_8';  
In the above query what it's going to return?

Finds any values in a five-digit number that start with 5 and end with 8.

59. select \* from Tbl\_Emp where Cell\_No like '\_6%8';  
In the above query what it's going to return?

Finds any values that have a 6 in the second position and end with a 8

**Note:** - The percent sign (%), the underscore (\_) are called Wildcard operators. The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

60. What is a Schema?

A schema is a collection of database objects (tables) associated with one particular database username. This username is called the schema owner, or the owner of the related group of objects. You may have one or multiple schemas in a database. The user is only associated with the schema of the same name and often the terms will be used interchangeably.

## 60. What is the difference between database and schema?

A database is the main container it contains the data and log files, and all the schemas within it. You always back up a database it is a discrete unit on its own.

Schemas are like folders within a database, and are mainly used to group logical objects together, which leads to ease of setting permissions by schema.

## 61. How can we get the list of system tables in a database?

```
Select*from sys.objects where type='s'
```

## 62. How can we get the list of user tables in a database?

```
Select*from sys.objects where type='u'
```

## 63. How can we get the list of Stored procedures in a database?

```
Select* from sys.objects where type='s'
```

## 64. How can we get the list of scalar functions in a database?

```
Select* from sys.objects where type='s'
```

## 65. How can we get the list of table valued functions in a database?

```
Select * from sys.objects where type='tf'
```

## 66. How can we get the list of triggers in a database?

```
Select * from sys.objects where type='tr'
```

## 66. Query to get list of Views?

```
select*from INFORMATION_SCHEMA.VIEWS;
```

## 67. Query to get list of all databases?

```
Select *from sys.databases;
```

## 68. Query to get list of all databases?

```
Select *from sys.databases;
```

---



69. Query to get list of system users?

Select \*from sysusers;

70. Query to get list of system logins?

Select \* from syslogins;

71. Query to get list of system logins?

Select \*from syslogins;

72. Query to get user?

Select CURRENT\_USER as Username;

73. Query to get the original login?

Select ORIGINAL\_LOGIN() as Original\_user;

74. Query to get the session user?

Select SESSION\_USER as 'Session\_user';

75. Query to get the system user?

Select SYSTEM\_USER as 'System\_user';

76. Query to get the database name?

Select db\_name()

77. Does views occupy memory?

No.

78. How can we cancel a transaction?

Using Rollback Transaction.

79. How to know your SQL Server version?

select @@version;

80. Can we call a trigger in store procedure?

We cannot call trigger explicitly from a store procedure as it is also a special kind of stored procedure. A Trigger will fire automatically on the happening of an event like before or after insert, update, delete.

---

## 81. What are the SQL Commands?

The SQL commands are used to interact with the database.

The commands can be classified into groups based on their nature:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Retrieval Language (DRL)
- Transaction Control Language (TCL)
- Data Control Language (DCL)

Command	Command	Description
<b>DDL</b>	CREATE	Creates a new table, a view of a table, or other object in database
	ALTER	Modifies an existing database object, such as a table.
	DROP	Deletes an entire table, a view of a table or other object in the database.
<b>DML</b>	INSERT	Creates a record
	UPDATE	Modifies records
	DELETE	Deletes records in the table
<b>DRL</b>	SELECT	Retrieves certain records from one or more tables
<b>TCL</b>	commit	It commits the transaction
	rollback	Cancelling the transaction to particular point (Save point)
	savepoint	Save the transaction gives a name
<b>DCL</b>	GRANT	Gives a privilege to user
	REVOKE	Takes back privileges granted from user

82. What are the difference between Primarykey, Unique key & foreign key's?

Primary Key	Unique key	Foregin key
Primary key cannot have a NULL value.	Unique Constraint may have a one NULL value.	Foreign key can accept multiple null values.
Each table can have only one primary key.	Each table can have more than one Unique Constraint.	We can have more than one foreign key in a table.
Primary key can be related to other tables as a Foreign Key.	Unique Constraint can not be related with another table's as a Foreign Key.	Foreign key is a field in the table that is Primary key in another table.
We can generate ID automatically with the help of Auto Increment field. Primary key supports Auto Increment value.	Unique Constraint doesn't supports Auto Increment value.	Applicable only when it's a applied to the parent table.
By default, Primary key is clustered index, and the data in database table is physically organized in the sequence of clustered index.	By default, Unique key is a unique non-clustered index.	Foreign keys do not automatically create an index, clustered or non-clustered. You must manually create an index on foreign keys.

83. What is Composite Key?

When multiple fields are used as a primary key, they are called a "composite key". Composite primary key individual values are accepted duplicate values but duplicate combination should not be repeated. It can be defined on max 16 columns only. It is defined at end of the table definition.

Sometimes more than one attributes are needed to uniquely identify an entity. A primary key that is made by the combination of more than one attribute is known as a composite key. Columns that make up the composite key can be of different data types.

84. What is Candidate key?

Any column(s) that can guarantee uniqueness is called a candidate key.

### 85. What is Alternate Key?

If a table has more than one candidate key, one of them will become the primary key and rest of all are called alternate keys.

Alternate key is a secondary key. In simple words, you can say that any of the candidate key which is not part of primary key is called an alternate key. So when we talk about alternate key, the column may not be primary key but still it is a unique key in the column.

### 86. What is the difference among NULL value, zero and blank space?

A NULL value is not same as zero or a blank space. A NULL value is a value which is 'unavailable, unassigned, unknown or not applicable'. On the other hand, zero is a number and blank space is treated as a character.

### 87. What is the difference between BETWEEN and IN condition operators?

The BETWEEN operator is used to display rows based on a range of values. The IN condition operator is used to check for values contained in a specific set of values.

### 88. What are scheduled tasks in SQL Server?

Scheduled tasks or jobs are used to automate processes that can be run on a scheduled time at a regular interval. This scheduling of tasks helps to reduce human intervention during night time and feed can be done at a particular time. User can also order the tasks in which it has to be generated.

### 89. How exceptions can be handled in SQL Server Programming?

Exceptions are handled using TRY--CATCH constructs and it is handles by writing scripts inside the TRY block and error handling in the CATCH block.

### 90. What is the purpose of FLOOR function?

FLOOR function is used to round up a non-integer value to the previous least integer.

Example: `select FLOOR(6.7);`  
returns --> 6

---

**91. Can we check locks in database? If so, how can we do this lock check?**

Yes, we can check locks in the database. It can be achieved by using in-built stored procedure called ***sp\_lock***.

**92. What is the use of SIGN function?**

SIGN function is used to determine whether the number specified is Positive, Negative and Zero. This will return +1, -1 or 0.

Example:

Select SIGN(-123); --- returns -1

Select SIGN(123); --- returns 1

Select SIGN(0); --- returns 0

**93. What is the difference between UNION and UNION ALL?**

UNION: To select related information from two tables UNION command is used. It is similar to JOIN command.

UNION ALL: The UNION ALL command is equal to the UNION command, except that UNION ALL selects all values. It will not remove duplicate rows, instead it will retrieve all rows from all tables.

**94. How Global temporary tables are represented and its scope?**

Global temporary tables are represented with ## before the table name. Scope will be outside the session whereas local temporary tables are inside the session.

**95. What are the differences between Stored Procedure and the dynamic SQL?**

Stored Procedure is a set of statements which is stored in a compiled form. Dynamic SQL is a set of statements that dynamically constructed at runtime and it will not be stored in a Database and it simply execute during run time.

**96. What is the command used to get the version of SQL Server?**

Select SERVERPROPERTY('productversion')

---

**97. What is UPDATE\_STATISTICS command?**

UPDATE\_STATISTICS command is used to update the indexes on the tables when there is a large amount of deletions or modifications or bulk copy occurred in indexes.

**98. What is the use of SET NOCOUNT ON/OFF statement?**

By default, NOCOUNT is set to OFF and it returns number of records got affected whenever the command is getting executed. If the user doesn't want to display the number of records affected, it can be explicitly set to ON- (SET NOCOUNT ON).

**99. Which SQL server table is used to hold the stored procedure scripts?**

Sys.all\_SQL\_Modules is a SQL Server table used to store the script of stored procedure. Name of the stored procedure is saved in the table called Sys.Procedures.

```
Select * from Sys.procedures;
```

```
Select * from sys.all_sql_modules;
```

**100. What are Magic Tables in SQL Server?**

Magic tables are invisible tables or virtual tables. Inserted and Deleted tables are created when the trigger is fired for any DML command. Those tables are called Magic Tables in SQL Server. These magic tables are used inside the triggers for data transaction.

**101. What is the use of =,==,=== operators?**

= is used to assign one value or variable to another variable. == is used for comparing two strings or numbers. === is used to compare only string with the string and number with numbers.

**102. What is ISNULL() operator?**

ISNULL() function is used to check whether value given is NULL or not NULL in sql server. This function also provides to replace a value with the NULL.

Example: `select eid,name,isnull(dept_id,100) from Emp_tbl`

---

### 103. Where are SQL Server user names and passwords stored in SQL Server?

User Names and Passwords are stored in sys.server\_principals and sys.sql\_logins. But passwords are not stored in normal text.

#### Example:

```
Select * from sys.server_principals;
```

```
Select * from sys.sql_logins;
```

### 104. What is the use of @@SPID?

A @@SPID returns the session ID of the current user process.

Example: `select @@SPID`

### 105. What is the difference between GETDATE and SYSDATETIME?

Both are same but GETDATE can give time till milliseconds and SYSDATETIME can give precision till nanoseconds. SYSDATE TIME is more accurate than GETDATE.

Example: `select GETDATE() as GetDateResult, SYSDATETIME() as SysDateResult;`

### 106. Which command is used for user defined error messages?

RAISEERROR is the command used to generate and initiates error processing for a given session. Those user defined messages are stored in sys.messages table. It has the details such as message id, severity and error messages etc.,

Example: `select * from sys.messages;`

### 107. What is Filtered Index?

Filtered Index is used to filter some portion of rows in a table to improve query performance, index maintenance and reduces index storage costs. When the index is created with WHERE clause, then it is called Filtered Index.

---

### 108. What is the difference between a HAVING CLAUSE and a WHERE CLAUSE?

The difference is that HAVING can be used only with the SELECT statement. HAVING is typically used in a GROUP BY clause. When GROUP BY is not used, HAVING behaves like a WHERE clause. Having Clause is basically used only with the GROUP BY function in a query whereas WHERE Clause is applied to each row before they are part of the GROUP BY function in a query.

### 109. What is Log Shipping?

Log shipping is the process of automating the backup of database and transaction log files on a production SQL server, and then restoring them onto a standby server. Enterprise Editions only supports log shipping. In log shipping the transactional log file from one server is automatically updated into the backup database on the other server. If one server fails, the other server will have the same db and can be used this as the Disaster Recovery plan. The key feature of log shipping is that it will automatically backup transaction logs throughout the day and automatically restore them on the standby server at defined interval.

### 110. What is @@Rowcount, @@Error and @@Identity?

**@@Rowcount** is used to display the number of rows affected by last SQL statement.

**@@Error** displays the error number for the last SQL statement executed. The value is zero, if there is no error.

**@@identity** returns the last inserted identity value.

#### **Example**

```
DECLARE @ErrorCode INT
DECLARE @RowsAffected INT
select 1/0
SELECT @ErrorCode = @@ERROR, @RowsAffected = @@ROWCOUNT
IF(@ErrorCode <> 0)
Print 'Error Occured'
ELSE IF(@RowsAffected = 0)
Print 'No Rows Affected'
```



### 111. What is a Linked Server?

When we want to query on remote database server along with the local database server then we can add the remote SQL server to local SQL server in a same group using the concept called Linked Server.

We can query on both servers using T-SQL. We can use stored Procedure `sp_addlinkedserver`, `sp_addlinkedsrvlogin` to add new Linked Server. By using Linked Server we can execute the SQL statements in clean and easy way to retrieve, join and combine remote data with local data.

### 112. Can SQL Servers linked to other servers like Oracle?

SQL Server can be linked to any server provided it has OLE-DB provider from Microsoft to allow a link. E.g. Oracle has an OLE -DB provider for oracle that Microsoft provides to add it as linked server to SQL Server group.

### 113. What is BCP? When does it used?

BulkCopy is a tool used to copy huge amount of data from tables and views. BCP does not copy the structures same as source to destination. BULK INSERT command helps to import a data file into a database table or view in a user-specified format.

### 114. What do you mean by ACID?

ACID (Atomicity Consistency Isolation Durability) is a quality sought after in a reliable database.

Here's the relevance of each quality:

Atomicity: It is an all-or-none proposition.

Consistency: It guarantees that your database is never left by a transaction in a half-finished state.

Isolation: It keeps transactions separated from each other until they're finished.

Durability: It ensures that the database keeps a track of pending changes in a way that the server can recover from an abnormal termination.

---

**115. What is a Database Lock?**

Database lock tells a transaction, if the data item in questions is currently being used by other transactions.

**116. Explain different types of Locks in SQL Server.**

There are 3 kinds of locks in SQL Server:

Shared locks: They are used for operations which do not allow any change or update of data. For e.g. SELECT

Update locks: They are used when SQL Server wants to modify a page. The update page lock is then promoted to an exclusive page lock before actually making the changes.

Exclusive locks: They are used for the data modification operations. For e.g. UPDATE, INSERT, or DELETE.

**117. What is a materialized view?**

Materialized views are also a view but are disk based. Materialized views get updates on specific duration, based upon the interval specified in the query definition. We can index materialized view.

**118. What is difference between View and Materialized view?**

View	Materialized view
View result set doesn't save anywhere on disk and executes the query definition whenever they are called.	Materialized view is similar to regular views but the output of select query has been saved to a table.
View shows the latest data all the time.	Materialized view only shows the fresh data after its result table is updated either by setting a schedule or based on the change in the underlying tables.
The performance of the view depends on how good the selected statement the view has. If the select statement has too many joins then it the view will perform poorly.	While in the case of materialized view, we are querying a table, which may also be indexed, that increase its performance.

**119. What is a WITH (NOLOCK)?**

WITH (NOLOCK) is used to unlock the data which is locked by the transaction that is not yet committed. This command is used before SELECT statement.

When the transaction is committed or rolled back then there is no need to use NOLOCK function because the data is already released by the committed transaction.

**Example:**

```
SELECT * FROM EmpDetails WITH (NOLOCK)
```

**120. What is difference between stored procedure and user defined function?**

Stored Procedure	User defined function
Stored procedure can return output parameters.	User defined functions do not return output parameters
Stored procedure when an error occurs the execution will ignore the error and jumps to the next statement.	The execution of User defined function will be stopped if any error occurred in it.
It is possible to change the data in the table by using stored procedure.	It is not possible to change the table data with the use of User defined functions

**Another Example on Inner join with 3 Tables:-****Table 1:-**

Create table Country

```
(Country_id int primary key, Country_Name Varchar (20));
```

**Table 2:-**

Create table State

```
(State_Id int primary key, state_Name varchar (20), country_id int  
foreignkey references country (Country_ID));
```

**Table 3:-**

Create table City

```
(City_Id int primary key, city_name varchar (20), State_id int foreign key  
references State (State_id));
```

**Query to fetch:-**

```
Select c.city_id, s.state_name, co.country_name from country co inner  
join state s on co.country_id=s.Country_id inner join city c on  
s.state_id=c.state_id;
```

***VENKAT:***  
***BIGDATA***  
***9866282929***