

MICROSOFT
SQL SERVER NOTES-2014

INTRODUCTION TO DBMS

Why DBMS: Human needs have increased tremendously. Now people are doing much more composite tasks than ever before. The society has become very complex. a person has to work with huge amount of information every day. In order to work with the enormous information, we must have a system where we can store, manipulate and share the information all over the world. It is one of the core reasons for introducing Database Management Systems (DBMS) as well as Relational Database Management Systems (RDBMS) now-a-days.

So, one thing is clear to us that we store and manipulate data / information into a database, where the database contains various types of tables for storing various types of data / information.

Data:

- Whatever we are inputting from the keyboard is known as Data. It can also be called as RAWFACTS / FIGURES
- Data never provides any meaning for us

Information:

- Processed Data is known as Information
- Information always gives meaning for us

Database:

- Collection of information belongs to a particular topic (an organization) written in a predetermined manner stored at a particular place is called as database.

DBMS (Data Base Management System):

- It is software which is present inside the database, which can maintain and manage the data within the database.

Types of DBMS:

1) FMS / FDMS (File Management System /File Management Database System):

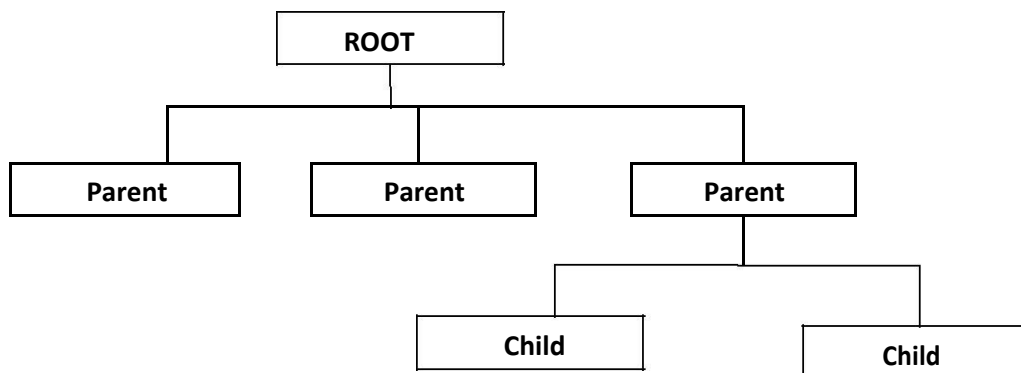
- This is first model released into the market in 1950"s. In this model there is always arranged in a continue stream of character (or) in a sequential fashion (or) manner.

Disadvantages:

- The Main disadvantage of this model is whenever we need to retrieve any data we have to start the searching from the beginning of the file so, it automatically leads to increases the searching time.
- Costly in maintains
- Required more man power
- There is no security

2) HMS/HDMS (Hierarchy Management System / Hierarchy Database Management System):

- This model was developed by IBM in 1960"s, When they developed a project called IMS (Information Management System)
- In this model data is always arranged in the form of a tree structure in different levels
- The top level can be called as root. The 2nd , 3rd can be called as parent and child levels respectively
- The main advantage of this model is we can easily retrieve the value without wasting much time



Drawback:

- Only one person can share the database simultaneously
- No security

3) NDBMS (Network Database Management System):

- This model was developed by IBM in 1969, when developing a project is called IMS (Information Management System)
- This model was developed on the basis of an Operating System called MULTICS (Multiplex Information Computing System)
- The main advantage of this model is more than one person can share the database concurrently (Simultaneously)

Disadvantage:

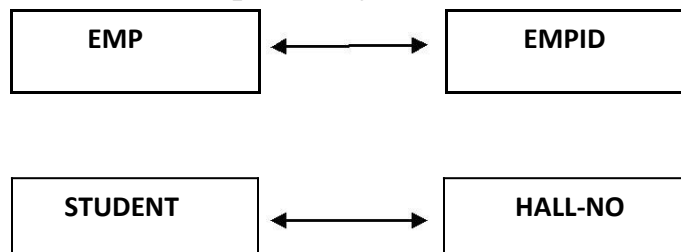
- There is no proper security for the centralized database
- Redundancy of the database is increased
- It occupies lot of memory and it leads to decrease system performance and increase the inconsistency.

4) RDMS (Relational Database Management System):

- This model was developed by a German scientist Mr. EF.CODD in 1970
- Here relation can be defined as commonness between objects these relations are classified into 3 types
 - One to One relation
 - One to Many relation / Many to One relation
 - Many to Many relation

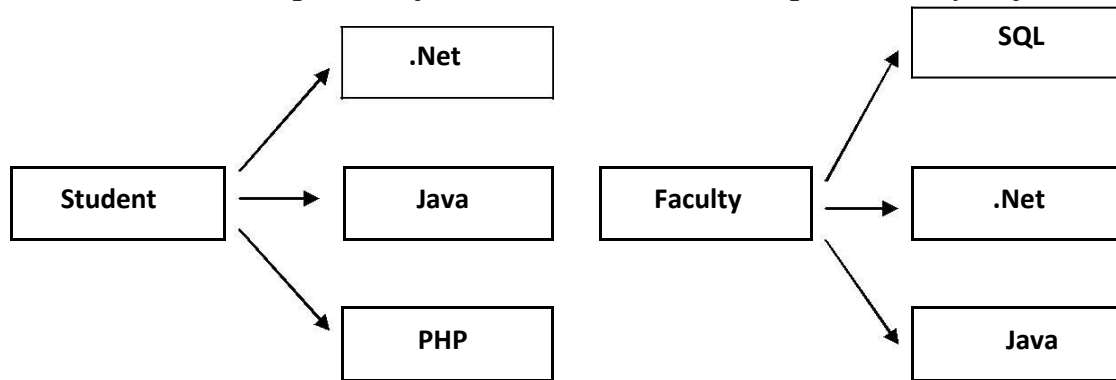
One – One relationship:

- In this relationship one object can have a relationship with another object



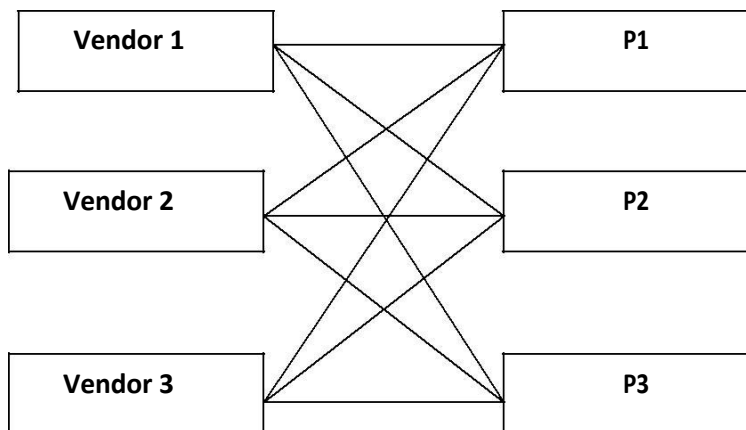
One - Many relationships:

- In this relationship one object can have a relationship with many objects



Many – Many relationship:

- In this relationship many vendors (or) many objects can have the relationship with many other objects



- All the above relationships can be called as “Degree of Relationships”
- This model was developed on the basis of a mathematical concept can be called as “Relation Algebra” (i.e. sets & Relations)

CODD RULES: E.F. Codd, the famous mathematician has introduced 12 rules for the relational model for databases commonly known as **Codd's rules**. The rules mainly define what is required for a DBMS for it to be considered relational i.e. an RDBMS. The rules are as follows:-

Rule 0: Foundation Rule

A relational database management system should be capable of using its relational facilities (exclusively) to manage the database.

Rule 1: Information Rule

All information in the database is to be represented in one and only one way. This is achieved by values in column positions within rows of tables.

Rule 2: Guaranteed Access Rule

All data must be accessible with no ambiguity, that is, Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

Rule 3: Systematic treatment of null values

Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

Rule 4: Dynamic On-line Catalog Based on the Relational Model

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data. The authorized users can access the database structure by using common language i.e. SQL.

Rule 5: Comprehensive Data Sublanguage Rule

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:

- a. data definition
- b. view definition
- c. data manipulation (interactive and by program)
- d. integrity constraints
- e. authorization
- f. Transaction boundaries (begin, commit, and rollback).

Rule 6: View Updating Rule

All views that are theoretically updateable are also updateable by the system.

Rule 7: High-level Insert, Update, and Delete

The system is able to insert, update and delete operations fully. It can also perform the operations on multiple rows simultaneously.

Rule 8: Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

Rule 9: Logical Data Independence

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

Rule 10: Integrity Independence

Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

Rule 11: Distribution Independence

The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.

Rule 12: No subversion Rule

If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.



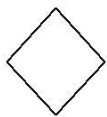

Note that based on these rules there is no fully relational database management system available today. In particular rules 6, 9, 10, 11 and 12 are difficult to satisfy.

Features of RDBMS

- In this model data should be stored in the form of tables
- A table can be defined as collection of rows & columns
- The horizontal lines are known as rows/ records / tuples
- The vertical lines are known as columns / fields / Attributes
- The intersection of rows & columns is known as cell
- A cell is a place where we can store our actual data
- The other name of table can be called as “Entity”
- It will provide high level security to database information
- Avoiding data redundancy problems
- Accessing the data from the table is not take much time
- When we define the column in the table user no need to follow any specific order

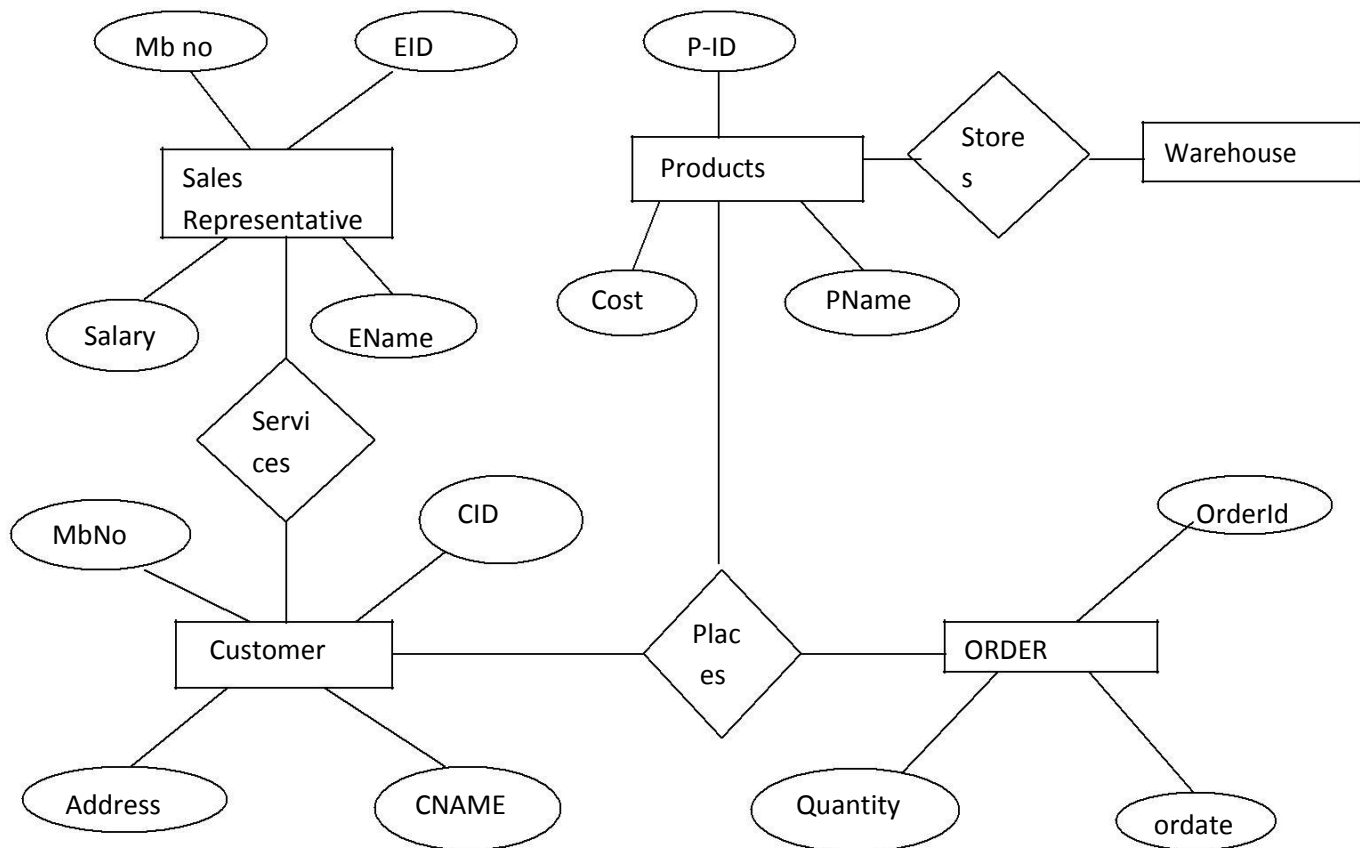
ER Diagram (Entity & Relationship diagram):

This is the pictorial representation of Manual database. This concept was developed by a US Scientist Mr.Chen

- Whenever we design ERD's the user has to follow the following symbols
 -  Rectangle box represent Entity
 -  Oval represent Attribute
 -  Diamond represents Relationship Name
 -  Arrow represent Connection



Draw an ER-Diagram to represent the relationship between sales representative & the customer



INTRODUCTION TO SQL SERVER

SQL SERVER: SQL Server is an RDBMS product which was designed and developed by Microsoft Company.

SQL Server will provide Graphical User Interface (GUI) facilities it means that user can interact with the database using icons without remember any commands.

SQL Server will run only windows operating system i.e. it is a platform dependent.

The first version of SQL SERVER is 1.0 was released in 1989 and up to now 11.0 versions are available. Those are...

| VERSIONS | YEAR | RELEASE NAME | CODE NAME |
|----------|------|--------------------|-------------|
| 1.0 | 1989 | SQL Server 1.0 | - |
| 1.1 | 1991 | SQL Server 1.1 | - |
| 4.21 | 1993 | SQL Server 4.21 | SQLNT |
| 6.0 | 1995 | SQL Server 6.0 | SQL95 |
| 6.5 | 1996 | SQL Server 6.5 | Hydra |
| 7.0 | 1998 | SQL Server 7.0 | Sphinx |
| 8.0 | 2000 | SQL Server 2000 | Shiloh |
| 9.0 | 2005 | SQL Server 2005 | Yukon |
| 10.0 | 2008 | SQL Server 2008 | Katmai |
| 10.5 | 2010 | SQL Server 2008 R2 | Kilimanjaro |
| 11.0 | 2012 | SQL Server 2012 | Denali |
| 12.0 | 2014 | SQL Server 2014 | SQL14 |

Working with SQL SERVER: SQL Server is a collection of databases where database is a collection of various objects like Tables, Views, Procedures and Functions etc.

To work on SQL Server we use SQL Server Management Studio. It is a tool will provide Command Based Environment and GUI Based Environment to perform operations in SQL server. If we connect to server it shows a window with

- Server Type
- Server Name
- Server Authentication
- Username & Password

Server Type: SQL server contains five types of servers those are

- **Database Engine:** The Database Engine is the core service for storing, processing, and securing data (or) it is used to store, manage and to access the data from the database.
- **Analysis Services (SSAS):** It is used for data warehouse it will show the data in three dimensions (Rows, Columns and New dimension).
- **Reporting Services (SSRS):** It is a reporting tool used to generate reports in various formats such as creating interactive, tabular, graphical,

multidimensional, or XML-based data sources. Reports can include rich data visualization, including charts, maps etc.

- **Integration Services(SSIS):** It is used to convert tables from relational database to another relational database for e.g. If we want to convert SQL Server tables to ORACLE tables or My SQL tables then will be used.
- **SQL Server Compact Edition:** It is used to develop mobile application or mobile software.

Server Authentication: We have two types of authentications are

Windows Authentication: Windows Authentication work on the user admin and when we work on window authentication there is no required user name and password because operating system will generate User Id and Password by default.

SQL Server Authentication: SQL Server will work on the current user and when we work on SQL Server authentication then user should enter User Id and Password (These User ID and Password will give at the time of SQL Server installation).

Step To Connect To SQL SERVER:

Go to start → Go to programs → Go to Microsoft SQL Server 2008 R2/12 → Click on SQL server management studio → Click on connect button.

Object Explorer Window: This window contain Database, Security, Server Objects, Replication and Management options.

SQL Server contains two types of databases these are

- **System Database:** The system database include the following four databases
 - **Master:** It is used to manage system level information of SQL server.
 - **Model:** It is used as a template for all new creating databases in SQL Server.

- **Msdb:** It is used to store the alerts and job information contains the SQL commands which are executed by user.
- **Tempdb:** When ever SQL server is started tempdb will be created in SQL server. It is used to store temporary tables once we restart the server the tempdb database is destroyed.
- **User Database:** These databases are created and manage by the user for storing their objects like tables, views, procedure etc.

Steps to Create User Database:

Go to open SQL server management studio → Click on Connect button to connect server
 → Go to Object Explorer window → Select Database and click on right mouse button
 → Click on new database option → Type database name in database name textbox control
 → Click on Ok button

- Whenever we create a database on SQL Server, it will generate two database files are

Primary Data file: It contain the start up information of the database and used to store database objects like tables, views .This file will saved with an extension .mdf(Master Data file).

Log File: This file contains transaction query information will saved with an extension .Ldf (Log Data file).

Root Location for .mdf and .ldf files:

C:\Program Files\Microsoft SQL
 Server\MSSQL10.MSSQLSERVER\MSSQL\DATA

Data Types in SQL Server: A data type is an attribute that specifies what types of data enter by the user such as integer, character, decimal, date time etc.



Integer Data Types



Decimal Data Types



Money (or) Currency Data Types



Date & Time Data Types



Character Data Types



Binary Data Types

Integer data type: It will allows integer values only such as EID, SID etc

| Data Type | Range | Stored Memory |
|-----------|---|---------------|
| Tiny Int | 0-255 | 1 byte |
| SmallInt | -32768 to 32767 | 2 bytes |
| Int | -2,147,483,648 to 2,147,483,647 | 4 bytes |
| Bigint | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 8 bytes |

Decimal Data Types: These data types will allow only decimal numbers and it can divide into two types but both are same.

- Decimal (P,S) -----> P=Precision & D= Scale
- Numeric (P,S)

Precision: It allows the total number of decimal digits i.e. both left and right side of the decimal point. The default precision is 18 and maximum 38.

Ex of Precision: 3457.78543 ----- Precision =9

Scale: It allows right side digits of decimal point only. The default value of scale is 0.

Ex of Scale: 3457.78543 ----- Scale = 5

| Precision | Stored Memory |
|-----------|---------------|
| 1-9 | 5 bytes |
| 10-19 | 9 bytes |
| 20-28 | 13 bytes |
| 29-38 | 17 bytes |

Money Data Type: This data type will allow currency values and it contains two types these are

| Data Type | Range | Stored Memory |
|-------------|---|---------------|
| Small money | -214,748,3648 To 214,748,3647 | 4 bytes |
| Money | - 922,337,203,685,477,5808 to 922,337,203,685,477,5807 | 8 bytes |

Date and Time Data Type: These data types are defined a particular date and time of the day.

Date: It defines date only the default format of date data type is “yy/mm/dd”

Time: It defines time of the day the default format is “hh/mm/ss.ms”

Date & Time: This will allow both date and time of the day.

“Yy/mm/dd hh/mm/ss.ms”

Character Data Types:

It allows to enter character values and these are classified into six types.

- **Char (n):** It is a fixed length data type, store the string values in non-Unicode manner i.e.it will take 1 char per 1 byte.
The maximum length of char data type is from 1-8000 bytes
- **Varchar (n/max):** It is a variable length data type, store the string values in non-Unicode manner i.e.it will take 1 char per 1 byte.
The maximum length of Varchar data type is from 1-8000 bytes
- **Text:** It is same as Varchar(max) data type
- **Nchar (n):** It is a fixed length data type, store the string values in Unicode manner i.e.it will take 1 char per 2 bytes.
The maximum length of char data type is from 1-4000 bytes
- **Nvarchar (n/max):** It is a variable length data type, store the string values in Unicode manner i.e.it will take 1 char per 2 byte.
The maximum length of char data type is from 1-4000 bytes
- **Ntext:** It is same as Nvarchar (max) data type .

Binary Data Type: Binary data types are used to store images, videos and audio data. These can be divided into the following types

- **Binary (n):** It is a fixed length data type. The maximum length of binary data type is 1-8000 bytes.
- **Varbinary (n/max):** It is a variable length data type. The maximum length of binary data type is 1-8000 bytes.
- **Image:** it is same as a Varbinary (max)

Note: Instead of text, Ntext and image data types are using Varchar (max), Nvarchar (max) and Varbinary (max) data types in latest versions of Microsoft SQL server.

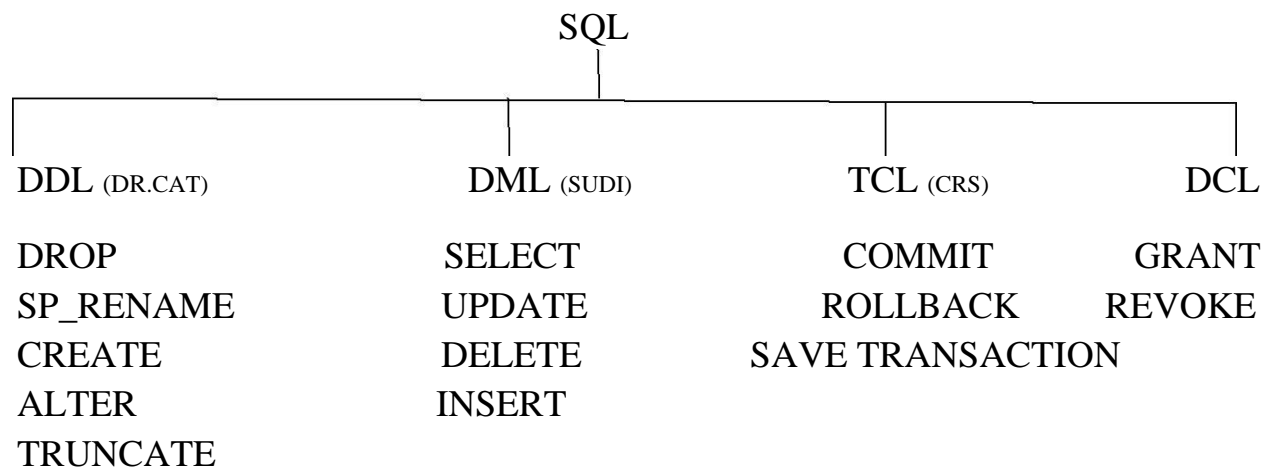
STRUCTURE QUERY LANGUAGE:

It is a non procedural language which is used to communicate with any database such as Oracle, sqlserver etc.

- This Language was developed by the German Scientist Mr. E.F.Codd in 1968
- ANSI (American National Standard Institute) approved this concept and in 1972 sql was released into the market

Features of SQL:

- SQL is not a case sensitive language it means that all the commands of Sql are not case sensitive
- Every command of sql should ends with a semicolon (;) (It is exemption for SQL Server)
- SQL can be pronounced as Sequel (Structured English Query Language)
- SQL can be called as Common Language Interface, which is used to communicate with any type of database
- SQL can be called as NLI (Natural Language Interface). It means that all the SQL Commands are almost similar to normal English language
- Structured query language is mainly divided into 4 sub languages
 1. DDL (Data Definition Language)
 2. DML (Data Manipulation Language)
 3. TCL (Transaction Control Language)
 4. DCL(Data Control Language)



DATA DEFINITION LANGUAGE

Data Definition Language: This is a 1st sub Language in SQL which is used to define the database objects such as table, view etc.

➤ This language contains five commands

1. Create
2. Alter
3. SP_Rename
4. Truncate
5. Drop

1. Create:

➤ This command is used to create the database objects within the database
Syntax: CREATE TABLE <TABLE NAME>

```
(COL 1 DATA TYPE (size),  
  COL2 DATA TYPE (size),  
  :  
  :  
  :  
  COLN DATA TYPE (size));
```

Ex: CREATE TABLE EMP (EID Int, ENAME Varchar (15), SAL
DECIMAL (6, 2));

Rules for Creating a Table:

- Table name must be unique under the database.
- Column names must be unique within the table.
- Never start table name with numeric or special characters except underscore“_”.
- Do not use space in table name if we want give space in table name then use underscore symbol only.
- Every object name should contain minimum one character and maximum 128 characters.
- The maximum no. of columns a table can have 1024 columns.

2. ALTER:

- This command is used to modify the structure of a table using this command, we can perform four different operations
 - Using this command we can increase (or) decrease the size of the data type & also we can change the data type from old data type to new data type
 - We can add a new column to the existing table
 - We can change the column name from old column name to new column name
 - We can remove the column from the existing table
- This command contains 4 sub commands
 1. ALTER- ALTER COLUMN
 2. ALTER- ADD
 3. SP_RENAME
 4. ALTER- DROP

a. ALTER-ALTER COLUMN:

- **Syntax:** ALTER TABLE <TABLE NAME> ALTER COLUMN <COLUMN NAME> DATA TYPE (SIZE)
- **Ex:** ALTER TABLE EMP ALTER COLUMN ENAME char (25);

b. ALTER-ADD:

- **Syntax:** ALTER TABLE <TABLE NAME> ADD <COLUMNNAME> DATA TYPE(size);
- **Ex:** ALTER TABLE EMP ADD DEPTNO int;

c. ALTER-DROP:

- **Syntax:** ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;
- **Ex:** ALTER TABLE EMP DROP COLUMN SAL;

d. SP_RENAME:

- **Syntax:** SP_RENAME „TABLENAME.OLDCOLUMN“,“NEW COLUMN NAME“,“COLUMN,;
- **Ex:** SP_RENAME „EMP.SAL“,“SALARY“,“COLUMN“

3. SP_RENAME:

- This command is used to change the table name from old table name to new table name
- **Syntax:** SP_Rename „old table name“,“ New table name“
- **Ex:** SP_Rename „EMP“,“EMP1“

4. TRUNCATE:

- This command is used for to delete all the records from existing table permanently
- **Syntax:** TRUNCATE TABLE <TABLE NAME>
- **Ex:** TRUNCATE TABLE EMP;

5. DROP:

- This command is used to remove the table permanently from the database
- **Syntax:** DROP TABLE <TABLE NAME>
- **Ex:** DROP TABLE EMP;

Note: SP_help: This command is used to see the structure of table

- **Syntax:** SP_help <table name>
- **Ex:** SP_help EMP

Note: Syntax to view tables in the current database.

- **select * from sysobjects where XTYPE='u'**

DATA MANIPULATING LANGUAGE

Data Manipulating Language: This is the 2nd sub language in SQL, which is used to manipulate the data within database. This Language contains 4 commands

1. Insert
2. Update
3. Delete
4. Select

1. INSERT:

- Using this command we can Insert the records into the existing table
- We can insert the records into the table in two methods
Explicit method
Implicit method

Explicit method:

- In this method user has to enter all the values into all the columns without anything omitting (or) left any column data
- **Syntax:** INSERT INTO <TABLE NAME> VALUES <VAL1, VAL2,VALN>;

(OR)

INSERT <TABLE NAME> VALUES <VAL1, VAL2, .VALN>;
(Here “INTO” Keyword is optional)

- **Ex1:** INSERT INTO EMP VALUES (101,“RAJ”,9500);
- **Ex2:** INSERT EMP VALUES (101,“RAJ”,9500);
1 Row(s) affected

Implicit method:

- In this method we can enter the values into the required columns in the table, so that user can omit (or) left some columns data while he enters the records into the table
- If the user omit any column data in the table then it automatically takes NULL
- **Syntax:** INSERT INTO <TABLE NAME> (COL1, COL2....COLN) VALUES (VAL1, VAL2... VALN);
- **Ex:** INSERT INTO EMP (EID, SAL) VALUES (106,9999);

2. UPDATE:

- This command is used to modify the data in the existing table
- By using this command we can modify all the records in the table & also specific records in the table (Using „where“ clause)
- **Syntax:** UPDATE <TABLE NAME> SET COL=VALUE;
- **Ex:** UPDATE EMP SET SAL=10000;

Syntax change for more than one data simultaneously

- **Syntax:** UPDATE <TABLE NAME> SET COL1=VALUE, COL2=VALUE.....COLN=VALUE;
- **Ex:** UPDATE EMP SET EID=007,SAL=10000;

3. DELETE:

- This command is used to delete the records from existing table
- Using this command we can delete all the records and also to delete specific record (by using „where“ clause)
- **Syntax:** DELETE FROM <TABLE NAME>
- **Ex:** DELETE FROM EMP;
10 row(s) affected

Difference between TRUNCATE and DELETE Command:

| SRNO | TRUNCATE | DELETE |
|-------------|--|--|
| 01 | It is a DDL command | It is a DML command |
| 02 | It is a permanent deletion | It is temporary deletion |
| 03 | Specific record deletion is not possible | We can delete the specific record |
| 04 | It doesn't support WHERE clause | It supports WHERE clause |
| 05 | We cannot Rollback the data | We can Rollback the data |
| 06 | Truncate will reset the identity Values | Delete will not reset the identity value |

4. SELECT:

- This command is used to retrieve the data from existing table.
- Using this command we can retrieve all the records & also specific records from existing table (by using „where“ clause)
- Using this command we can retrieve the data from the table in 3 ways
 - 1. Projection**
 - 2. Selection**
 - 3. Joins**
- **Syntax:** SELECT * FROM <TABLE NAME>
- **Ex:** SELECT * FROM EMP;
- * represents all columns

Projection:

- Retrieving the data from specific columns is known as Projection
- **Syntax:** SELECT COL1,COL2.....COLN FROM <TABLE NAME>
- **Ex:** SELECT EID,ENAME FROM EMP;

Selection:

- Retrieving the data based on some condition is called selection
- In SQL, whenever we need to check a condition, we need to use a special clause called „where“
- **Syntax:** SELECT * FROM <TABLENAME> WHERE (CONDITION);
- **Ex:** SELECT * FROM EMP WHERE EID=101;

WHERE CLAUSE:

- This clause is used to check the condition based on the condition, we can retrieve, update, delete specific records in the table
- So we can apply the where clause only in select, update & delete

Select Command With Where clause:

- **Syntax:** SELECT * FROM <TABLE NAME> WHERE <CONDITION>
- **Ex:** SELECT * FROM EMP WHERE EID=102;

Update Command With Where clause:

- **Syntax:** UPDATE <TABLE NAME> SET <COLUMN NAME>=VALUE WHERE (CONDITION);
- **Ex:** UPDATE EMP SET ENAME=„sai“ WHERE EID=102;

Delete Command With Where clause:

- **Syntax:** DELETE FROM <TABLE NAME>WHERE <CONDITION>
- **Ex:** DELETE FROM EMP WHERE EID=102;

ALIAS:

- ALIAS is a duplicate name (or) alternate name for the original column name (or) Table name (or) an expression name.
- **Column level Alias:**
- **Syntax:** SELECT COLUMN NAME AS “ALIAS NAME”,
COLUMN NAME AS “ALIAS NAME”,
:
:
COLUMN NAME AS “ALIAS NAME” FROM <TABLE NAME>;
- **EX:** SELECT EID AS “EMPLOYEE ID”, ENAME AS “EMPLOYEE NAME”, SAL AS “SALARY” FROM EMP;
- **NOTE:** In the above example the keyword „as“ is optional
- **EX:** SELECT EID “EMPLOYEE ID”, ENAME “EMPLOYEE NAME”, SAL “SALARY” FROM EMP;
- **NOTE:** In the above example quotations is also optional but there should not be space between column name
- **EX:** SELECT EID EMPLOYEEID, ENAME EMPLOYEEENAME, SAL SALARY FROM EMP;
- **Ex:** SELECT EID EMPLOYEEID, ENAME EMPLOYEEENAME, SAL SALARY, SAL*12 ANNUALSALARY FROM EMP;
- **EX:** SELECT EID EMPLOYEEID, ENAME EMPLOYEEENAME, SAL SALARY FROM EMP WHERE ANNUALSALARY > 115000
- In the above example returns the runtime error message invalid column name „annual salary“ because we cannot check the conditions on Alias name

IDENTITY: It is use to generate unique values in sequential order without user interaction. The default value of identity is Identity (1, 1).

Syntax: Identity (seed, increment)

Ex: CREATE TABLE EMP (EID INT IDENTITY (100, 1), ENAME VARCHAR (50));

Built In Functions(System Functions) IN SQL: SQL server

provide number of built in functions like mathematical functions, character functions, date and time functions, aggregative functions, conversion functions etc. these can be used to perform certain operations and return a value.

Syntax: **SELECT** <Function Name> [Expressions]

Mathematical Functions: These functions perform a calculation based on input values provided as arguments, and return a numeric value.

ABS (): Returns the absolute, positive value of the given numeric expression.

Ex: select ABS(-15)---- 15
select ABS(45)----- 45

CEILING (): Returns the smallest integer greater than, or equal to, the given numeric expression.

Ex: select ceiling(15.000)----15
select ceiling(15.0001)----16
select ceiling(-12.34)-----(-12)

FLOOR (): Returns the largest integer less than or equal to the given numeric expression.

Ex: select floor(15.000)---15
select floor(15.0001)----15
select floor(-12.34)----(-13)

SQUARE (): Returns the square of the given expression.

Ex: select SQUARE(5)---25

SQRT (): Returns the square root of the given expression.

Ex: select SQUARE(25)---5

POWER (n, m): Returns the power value of given expression

Ex: select POWER (2, 3) ----- 8

SIGN (): Returns the positive (+1), zero (0), or negative (-1) sign of the given expression.

Ex: select SIGN(42)-----1
select SIGN(0)-----0
select SIGN(-42)-----(-1)

PI (): Returns the constant value of PI.

Ex: select PI()-----3.14159265358979

LOG (): Returns the natural logarithm of the given expression.

Ex: select LOG(2)----- 0.693147180559945

LOG 10(): Returns the base-10 logarithm of the given expression.

Ex: select LOG10(10)----1

SIN (): Returns the trigonometric sine of the given angle (in radians) in an approximate numeric expression.

Ex: select SIN (0) -----0

COS (): A mathematic function that returns the trigonometric cosine of the given angle (in radians) in the given expression.

Ex: select COS (0) -----1

TAN (): Returns the tangent of the input expression.

Ex: select TAN (0) -----0

String Functions: These functions perform an operation on a string input value and return a string or numeric value.

ASCII (): Returns the ASCII code value of the leftmost character of a character expression.

Ex: Select ASCII („Z“) -----90

CHAR (): A string function that converts an **int** ASCII code to a character.

Ex: Select CHAR (90) -----Z

CHARINDEX (): Returns the starting position of the specified expression in a character string.

Ex: Select CHARINDEX (,S", "SUDHAKAR") -----1

LEFT (): Returns the left part of a character string with the specified number of characters.

Ex: Select LEFT (,SUDHAKAR", 5) ----SUDHA

RIGHT (): Returns the right part of a character string with the specified number of characters.

Ex: Select RIGHT (,SUDHAKAR", 3) -----KAR

LEN (): Returns the number of characters, rather than the number of bytes, of the given string expression.

Ex: Select LEN (,WELCOME") -----7

LOWER (): Returns a character expression after converting uppercase character data to lowercase.

Ex: Select LOWER (,SAI") -----sai

UPPER (): Returns a character expression with lowercase character data converted to uppercase.

Ex: Select UPPER (,sai") -----SAI

LTRIM (): Returns a character expression after removing leading blanks.

Ex: Select LTRIM (, HELLO") -----HELLO

RTRIM (): Returns a character string after truncating all trailing blanks.

Ex: Select RTRIM (,HELLO ,,) -----HELLO

REPLACE (): Replaces all occurrences of the second given string expression in the first string expression with a third expression.

Ex: Select REPLACE (,JACK AND JUE", ,J", ,BL") -----BLACK AND BLUE

REPLICATE (): Repeats a character expression for a specified number of times.

Ex: Select REPLICATE („SAI“, 3) -----SAISAI

REVERSE (): Returns the reverse of a character expression.

Ex: Select REVERSE („HELLO“) -----OLLEH

SPACE (): Returns a string of repeated spaces.

Ex: Select („SAI“+SPACE (50) +“SUDHAKAR“) -----SAI SUDHAKAR

SUBSTRING (expression, start, length): Returns a part of a string from expression from starting position, where length is no. of chars to be picked.

Ex: Select SUBSTRING („HELLO“, 1, 3) ----- HEL

Select SUBSTRING („HELLO“, 3, 3) ----- LLO

Date and Time Functions: These functions perform an operation on a date and time input value and return a string, numeric, or date and time value.

GETDATE (): Returns the current system date and time in the SQL Server standard internal format for date time values.

Ex: Select GETDATE () ----- 2014-02-15 15:35:22.670

DAY (): Returns an integer representing the day date part of the specified date.

Ex: Select DAY (get date ())

MONTH (): Returns an integer that represents the month part of a specified date.

Ex: Select MONTH (get date ())

YEAR (): Returns an integer that represents the year part of a specified date.

Ex: Select YEAR (get Date ())

GETUTCDATE (): Returns the date time value representing the current UTC time (Coordinated Universal Time).

Ex: Select GETUTCDATE ();

DATE NAME (): Returns a character string representing the specified date part of the specified date.

Ex: Select DATE NAME (DW, get date ())

DATE PART (): Returns an integer representing the specified date part of the specified date.

Ex: Select DATEPART (DD, get date ())

DATE ADD (): Returns a new date time value based on adding an interval to the specified date.

Ex: Select DATEADD (DD, 5, get date ())

DATE DIFF (): Returns the difference between the start and end dates in the give date part format.

Ex: Select DATEDIFF (MM, „2012-12-15“, get date ())

Conversion Functions: These functions are used to convert one data type to another. We have two conversion functions are CAST and CONVERT both provide similar functionality.

CAST (): Convert to one data type to another type.

Syntax: CAST (Expression as data type [size])

Ex: Select CAST (10.2587 as Int) -----10

CONVERT (): Convert function can be used to display date time data in different format.

Syntax: Convert (Data type [size], Expression, Style value)

Ex: Select Convert (Varchar (24), get date (), 113)

The table below represents the style values for date time or small date time conversion to character data:

| Sno | Value | Output | Standard |
|------------|--------------|-------------------------------------|------------------|
| - | 0 or 100 | mon dd yyyy hh:mi AM (or PM) | Default |
| 1 | 101 | mm/dd/yy | USA |
| 2 | 102 | yy.mm.dd | ANSI |
| 3 | 103 | dd/mm/yy | British/French |
| 4 | 104 | dd.mm.yy | German |
| 5 | 105 | dd-mm-yy | Italian |
| 6 | 106 | dd mon yy | |
| 7 | 107 | Mon dd, yy | |
| 8 | 108 | hh:mm:ss | |
| - | 9 or 109 | mon dd yyyy hh:mi:ss:mmmAM (or PM) | Default+millisec |
| 10 | 110 | mm-dd-yy | USA |
| 11 | 111 | yy/mm/dd | Japan |
| 12 | 112 | Yymmdd | ISO |
| - | 13 or 113 | dd mon yyyy hh:mi:ss:mmm (24h) | |
| 14 | 114 | hh:mi:ss:mmm (24h) | |
| - | 20 or 120 | yyyy-mm-dd hh:mi:ss (24h) | |
| - | 21 or 121 | yyyy-mm-dd hh:mi:ss:mmm (24h) | |
| - | 126 | yyyy-mm-ddThh:mi:ss:mmm (no spaces) | ISO8601 |
| - | 130 | dd mon yyyy hh:mi:ss:mmmAM | Hijiri |
| - | 131 | dd/mm/yy hh:mi:ss:mmmAM | Hijiri |

Aggregate functions/Group functions: Aggregate functions perform a calculation on a set of values and return a single value. Aggregate functions are often used with the GROUP BY clause of the SELECT statement.

SUM (): Returns the sum of all the values .Sum can be used with numeric columns only. Null values are ignored.

Ex: SELECT SUM (SALARY) FROM EMP

AVG (): Returns the average of the values in a group. Null values are ignored.

Ex: SELECT AVG (SALARY) FROM EMP

MAX (): Returns the maximum value in the expression.

Ex: SELECT MAX (SALARY) FROM EMP

MIN (): Returns the minimum value in the expression.

Ex: SELECT MIN (SALARY) FROM EMP

COUNT (): Returns the number of records in a table. This function again use in three ways.

1. **COUNT (*):** It Returns total number of records in a table **Ex:** SELECT COUNT (*) FROM EMP
2. **COUNT (Expression/Column name):** It returns number of records including duplicate values but not null vales.
Ex: SELECT COUNT (ENAME) FROM EMP
3. **COUNT (Distinct Column name):** It returns number of records without null and duplicate values.
Ex: SELECT COUNT (Distinct ENAME) FROM EMP

Distinct Key: If we use this key word on a column with in a query then it will retrieve the values of the column without duplicates.

OPERATORS IN SQL: Operator is a symbol which performs some specific operation on operands or expressions. These operators are classified into 6 types in SQL.

1. Assignment operator
2. Arithmetic operator
3. Comparison operator
4. Logical operator
5. Set operator

Assignment operator: Assignment operator contain only one operator is known as equal „=“ operator.

Ex1: Write a Query to display the employee details whose salary is equal to 10000

- `SELECT * FROM EMP WHERE SAL=10000`

Ex2: Write a query to change the deptno as „10“ whose employee id is 101

- `UPDATE EMP SET DEPTNO=10 WHERE EID=101`

Ex3: Write a query to delete a record whose employee id is 107

- `DELETE FROM EMP WHERE EID=107`

Arithmetic operator: Arithmetic operators perform mathematical operations on two expressions. The lists of arithmetic operators are + (Add), - Subtraction, * Multiplication, / (Divide) Division and % (Modulo) Returns the integer remainder of a division. For example, $12 \% 5 = 2$ because the remainder of 12 divided by 5 is 2.

Ex1: Select $100+250$

Select $245-400$

Select $20*20$

Select $25/5$

Select $37\%6$

Select 20/5+20/5

Select 35.50+20

Ex2: WAQ to find student TOTAL, AVERAGE AND CLASS OF a table

Step1: Create table student (Sid int, sname varchar (50), math's int, phy int, che int, total int, average int, class varchar (max))

Step2: Update student set total=maths+phy+che

Step3: Update student set average=total/3

Step4: Update student set class=

Case

When average>=60 then 'First class'

When average>=50 then 'second class'

When average>=40 then 'third class'

Else

'Fail'

End

CASE (): This function is used to execute list of conditions and returns a value.

Syntax: Case

<Condition 1>-----<Condition N>

Else

<Statement>

End

Comparison operators: Comparison operators test whether two expressions are the same. Comparison operators can be used on all expressions except expressions of the text, ntext, or image data types. The following table lists the Transact-SQL comparison operators are > (Greater Than), < (Less Than) [HYPERLINK "http://technet.microsoft.com/en-us/library/ms179873.aspx"](http://technet.microsoft.com/en-us/library/ms179873.aspx) ,>= (Greater Than or Equal To) ,<= (Less Than or Equal To) ,!= (Not Equal To),!< (Not Less Than),!> (Not Greater Than)

Examples:

- Select ename from EMP where salary<50000
- Update EMP set salary=1000 where salary>90000
- Update EMP set ename='joshitha' where salary<=25000
- Update EMP set salary=98000 where salary>=1000
- Select ename from Emp where salary !=98000
- Select ename from Emp where salary <98000
- Select ename from Emp where salary !=98000

Logical operator: Logical operators test for the truth of some condition. Logical operators, like comparison operators, return a Boolean data type with a value of TRUE or FALSE. Logical operators are AND , OR , NOT, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, IN, NOT IN, EXISTS,NOT EXISTS, ANY, ALL, SOME.

Examples:

- Select * from EMP where ename='siddhu' and salary=45000
- Select * from EMP where ename='joshitha' or salary=98000
- Select * from EMP where not ename='joshitha'
- Select * from EMP where salary between 10000 and 50000
- Update EMP set ename='SAI' where eid=101 and salary=25000

Queries Using 'Select' with 'where' clause:

- Write a Query to display the employee details whose salary is less than 10000
- SELECT * FROM EMP WHERE SAL<10000

➤ Write a Query to display the employee details whose salary is greater than or equal to 9000 and less than 15000

- **SELECT * FROM EMP WHERE SAL >= 9000 AND SAL <= 15000**
(OR)

- **SELECT * FROM EMP WHERE SAL BETWEEN 9000 AND 15000**

➤ Write a Query to display the employee details whose salary is not between 9000 and 15000

- **SELECT * FROM EMP WHERE SAL NOT BETWEEN 9000 AND 15000**

➤ Write a Query to display the employee details whose name starts with „r“

- **SELECT * FROM EMP WHERE ENAME LIKE „r%“**

➤ Write a Query to display the employee details whose name ends with „y“ •
SELECT * FROM EMP WHERE ENAME LIKE „%Y“

➤ Write a Query to display the employee details whose name contains the letter „a“

- **SELECT * FROM EMP WHERE ENAME LIKE „%A%“**

➤ Write a Query to display the employee details whose names contain only three letters

- **SELECT * FROM EMP WHERE ENAME LIKE „---“**

➤ Write a Query to display the employee details whose names contain „r“ and salary greater than 9000

- **SELECT * FROM EMP WHERE ENAME LIKE „%R%“ AND SAL > 9000**

➤ Write a Query to display the employee details whose greater than ram

- **SELECT * FROM EMP WHERE ENAME > „RAM“**

➤ Write a Query to display the employee details whose employee id starts with 1 and ends with 1

- **SELECT * FROM EMP WHERE EID LIKE „1%1“**

(SQL commands are not case sensitive and also data available in SQL also not case sensitive, in oracle Data available is case sensitive)

Queries using 'Update' with 'where' clause:

- Write a query to change the deptno as „10“ whose employee id is 101, 103, 107
 - UPDATE EMPSET DEPTNO=10 WHERE EID=101 OR EID=103 OR EID=107

- Write a query to change the deptno as 20 who does not have deptno •
UPDATE EMPSET DEPTNO=20 WHERE DEPTNO IS NULL

- Write a query to change the employee salaries as 12000 who are working under 10 dept and their names starts with „r“
 - UPDATE EMPSET SAL=12000 WHERE DEPTNO=10 AND ENAME LIKE „R%“

- Write a query to change the deptno as 30 whose second letter is „a“ •
UPDATE EMPSET DEPTNO=30 WHERE ENAME=“-A%“

- Write a query to change the employee salaries as 8500 who are working under 10 and 20 deptno
 - UPDATE EMPSET SAL=8500 WHERE DEPTNO=10 OR DEPTNO=20
(OR)
 - UPDATE EMPSET SAL=8500 WHERE DEPTNO IN(10,20)

- Write a query to change the employee salaries as 8500 who are not working under 10 and 20 deptno
 - UPDATE EMPSET SAL=8500 WHERE DEPTNO NOT IN (10,20)

- Write a query to change the employee salaries as 15000 and names ends with „m“ & working under 10 deptno
 - UPDATE EMPSET SAL=15000 WHERE ENAME=“%M“ AND DEPTNO=10

- Write a query to change the employee salaries as 5500 whose employee id ends with 4 and deptno starts with 2

- UPDATE EMPSET SAL=5500 WHERE EID LIKE „%4“ AND DEPTNO LIKE „2%“
- Write a query to change the employee salaries as 25000 whose salary less than 10000 and the name contains letter „a“ and working under dept 20
- UPDATE EMPSET SAL=25000 WHERE SAL<10000 AND ENAME LIKE „%A%“ AND DEPTNO IN (20)
- Write a query to change the employee salaries as 10000 whose salary is greater than or equal to 8500 and less than or equal to 9000
- UPDATE EMPSET SAL=10000 WHERE SAL BETWEEN 8500 AND 9000

Set Operators: Set operators combine results from two or more queries into a single result set. SQL Server provides the following set operators.

- UNION
- UNION ALL
- INTERSECT
- EXCEPT

To combine the results of two queries we need to follow the below basic rules.

- The number and the order of the columns must be the same in all queries.
- The data types must be compatible(Well-Matched)

Example:

```
CREATE TABLE EMP_HYD (EID INT, ENAME VARCHAR (50), SALARY MONEY)
```

```
CREATE TABLE EMP_CHENNAI (EID INT, ENAME VARCHAR (50))
```

EMP_HYD

| EID | ENAME | SALARY |
|-----|--------|----------|
| 101 | SAI | 25000.00 |
| 102 | SIDDHU | 32000.00 |
| 103 | KAMAL | 42000.00 |
| 104 | NEETHU | 63000.00 |

EMP_CHENNAI

EID ENAME

101 SAI

105 POOJA

106 JASMIN

UNION: it combines the result of two or more select statements into a single result set that includes all the records belongs to all queries except duplicate values.

Ex: Select Ename from EMP_HYD

Union

Select Ename from EMP_CHENNAI

OUTPUT: ENAME

JASMIN

KAMAL

NEETHU

POOJA

SAI

SIDDHU

UNION ALL: it is same as union but returns duplicate values

Ex: Select Ename from EMP_HYD

Union ALL

Select Ename from EMP_CHENNAI

OUTPUT: ENAME

| |
|--------|
| SAI |
| SIDDHU |
| KAMAL |
| NEETHU |
| SAI |
| POOJA |
| JASMIN |

INTERSECT: INTERSECT returns any distinct values that are common in left and right tables.

Ex: Select Ename from EMP_HYD

Intersect

Select Ename from EMP_CHENNAI

OUTPUT: ENAME SAI

EXCEPT: EXCEPT returns any distinct values from the left query that are not found on the right query.

Ex: Select Ename from EMP_HYD

Except

Select Ename from EMP_CHENNAI

| | |
|-----------------------|--------------|
| <u>OUTPUT:</u> | <u>ENAME</u> |
| | KAMAL |
| | NEETHU |
| | SIDDHU |

Ex: Select Ename from EMP_CHENNAI

Except

Select Ename from EMP_HYD

| | |
|-----------------------|--------------|
| <u>OUTPUT:</u> | <u>ENAME</u> |
| | JASMIN |
| | POOJA |

CLAUSES IN SQL: We can add these to a query for adding additional options like filtering the records, sorting records and grouping the records with in a table. These clauses contains the following clauses are,

WHERE: This clause is used for filter or restricts the records from the table.

Ex: SELECT * FROM EMP WHERE SAL=10000

ORDER BY: The order by clause is used to sort or arrange the data in ascending or descending order with in table. By default order by clause arrange or sort the data in ascending order only.

- If we want to arrange the records in a descending order then we use Desc keyword.
- We can apply order by clause on integer and string columns.

Ex: SELECT * FROM EMP ORDER BY EID (For Ascending Order)

Ex: SELECT * FROM EMP ORDER BY ENAME DESC (For Descending Order)

TOP N CLAUSE: This clause is used to fetch a top n number of records from a table.

Ex: SELECT TOP 3 * FROM EMP

Ex: UPDATE TOP 3 EMP SET ENAME="SAI"

Ex: DELETE TOP 3 FROM EMP

GROUP BY: Group by clause will use for to arrange similar data into groups. when we apply group by clause in the query then we use group functions like count(),sum(),max(),min(),avg().

If we use group by clause in the query, first the data in the table will be divided into different groups based on the columns and then execute the group function on each group to get the result.

Ex1: WAQ to find out the number of employees working in the organization

Sol: SELECT COUNT (*) FROM EMP

Ex2: WAQ to find out the number of employees working in each group in the organization.

Sol: SELECT DEPT, COUNT=COUNT (*) FROM EMP GROUP BY DEPT

Ex3: WAQ to find out the total salary of each department in the organization

Sol: SELECT DEPT, TOTALSALARY=SUM (SALARY) FROM EMP GROUP BY DEPT (Like this we can find max, min, avg salary in the organization)

HAVING CLAUSE: Having clause is also used for filtering and restricting the records in a table just like where clause.

Ex: WAQ to find out the number of employees in each department only if the count is greater than 3

Sol: SELECT DEPT, COUNT=COUNT (*) FROM EMP GROUP BY DEPT
HAVING COUNT (*) >3

Differences Between WHERE and HAVING Clause:

| WHERE | HAVING |
|--|--|
| WHERE clause is used to filter and restrict the records before grouping | HAVING clause is used to filter and restrict the records after grouping |
| If restriction column associated with A aggregative function then we cannot use WHERE clause there | But we can use HAVING clause at this situations |
| WHERE clause can apply without group by clause | HAVING clause cannot be applied without a group by clause |
| WHERE clause can be used for restricting individual rows | Where as HAVING clause is used along with group by clause to filter or restrict groups |

SYNONYM: synonym is database object which can be created as an “alias” for any object like table, view, procedure etc.

- If we apply any DML operations on synonym the same operations automatically effected to corresponding base table and vice versa.
- If we create a synonym, the synonym will be created on entire table. It is not possible to create the synonym on partial table.
- When we create synonym based on another the new synonym does not allow us to perform any DML operations because Synonym chaining is not allowed.
- Synonym will become invalid into two cases,
 1. When we drop the base table
 2. When we change the base table name

- On invalid synonym we cannot apply any DML operations and we cannot create synonym based on more than one table at a time.
- When we change the structure of the base table the corresponding synonym automatically reflected with same changes.
- But, if we change the structure of the synonym that is not reflected to the base table because we cannot change the structure of the synonym.

Syntax: Create synonym <synonym name> for <object name>

Ex: Create synonym synemp for employee

Syntax to drop a synonym: Drop synonym <synonym name>

Ex: Drop synonym synemp

Syntax to Creating a table from an existing table:

we can create a table from an existing table and maintain a copy of the actual table before manipulating the table.

Syntax: Select * into <New Table Name> from <Old Table Name>

Ex1: Select * into New_Emp from Employee

In this case it creates a table New_Emp by copying all the rows and columns of the Employee table.

Ex2: Select EID, ENAME into Test_Emp from Employee

In this case it creates a table Test_Emp with only the specified columns from the employee table.

Ex3: Select * into Dummy_Emp from employee where 1=2 In

this case it creates the Dummy table without any data in it.

Copying data from one existing table to another table:

We can copy the data from one table to another table by using a combination of insert and select statement as following

Syntax: Insert into <Dummy Table name> select * from <Table Name>

Ex: Insert into Dummy_Emp select * from Employee

Constraint in SQL

Why Constraint in SQL: Constraint is using to restrict the insertion of unwanted data in any columns. We can create constraints on single or multiple columns of any table. It maintains the data integrity i.e. accurate data or original data of the table. Data integrity rules fall into three categories:

- Entity integrity
- Referential integrity
- Domain integrity

Entity Integrity: Entity integrity ensures each row in a table is a uniquely identifiable entity. You can apply entity integrity to a table by specifying a PRIMARY KEY and UNIQUE KEY constraint.

Referential Integrity: Referential integrity ensures the relationships between tables remain preserved as data is inserted, deleted, and modified. You can apply referential integrity using a FOREIGN KEY constraint.

Domain Integrity: Domain integrity ensures the data values inside a database follow defined rules for values, range, and format. A database can enforce these rules using CHECK KEY constraints.

Types of constraints in SQL Server:-

1. Unique Key constraint.
2. Not Null constraint.
3. Check constraint
4. Primary key constraint.
5. Foreign Key constraint.

1. Unique Key:- Unique key constraint is use to make sure that there is no duplicate value in that column. Both unique key and primary key both enforces the uniqueness of column but there is one difference between them unique key constraint allow null value but primary key does not allow null value.

In a table we create one primary key but we can create more than one unique key in Sql Server.

Ex: create table **EMP** (EID int unique, ENAME varchar(50) unique, SALARY money);

2. Not null constraint: - Not null constraint is used to restrict the insertion of null value at that column but allow duplicate values.

Ex: create table **EMP** (EID int not null, ENAME varchar(50) not null, SALARY money);

3. Check Constraint: - This constraint is using to check value at the time of insertion like as salary of any employee is always greater than zero. So we can create a check constraint on employee table which is greater than zero.

Ex: create table **emp4** (eno int, ename varchar(50), age int check (age between 20 and 30))

4. Primary Key:- Primary key is a combination of unique and not null which does not allow duplicate as well as null values into a column. In a table we create one primary key only.

Ex: create table **emp** (EID int primary key, ENAME varchar(50), SALARY money)

5. Foreign Key: - One of the most important concepts in database is creating relationships between database tables. These relationships provide a mechanism for linking data stored in multiple tables and retrieving it in an efficient manner.

In order to create a link between two tables we must specify a foreign key in one table that references a column in another table.

Foreign key constraint is used for relating or binding two tables with each other and then verifies the existence of one table data in the other.

To impose a foreign key constraint we require the following things.

We require two tables for binding with each other and those two tables must have a common column for linking the tables.

Example:

To create Department Table (PARENT TABLE):-

```
create table Department(Deptno int primary key,DNAME  
varchar(50),LOCATION varchar(max))
```

Insert Records Into Department Table:

```
insert into Department values(10,'Sales','Chennai') insert  
into Department values(20,'Production','Mumbai') insert  
into Department values(30,'Finance','Delhi') insert into  
Department values(40,'Research','Hyderabad')
```

To create Employee Table(CHILD TABLE):-

```
create table Employee(EID int,ENAME varchar(50),SALARY money,Deptno  
int foreign key references Department(Deptno))
```

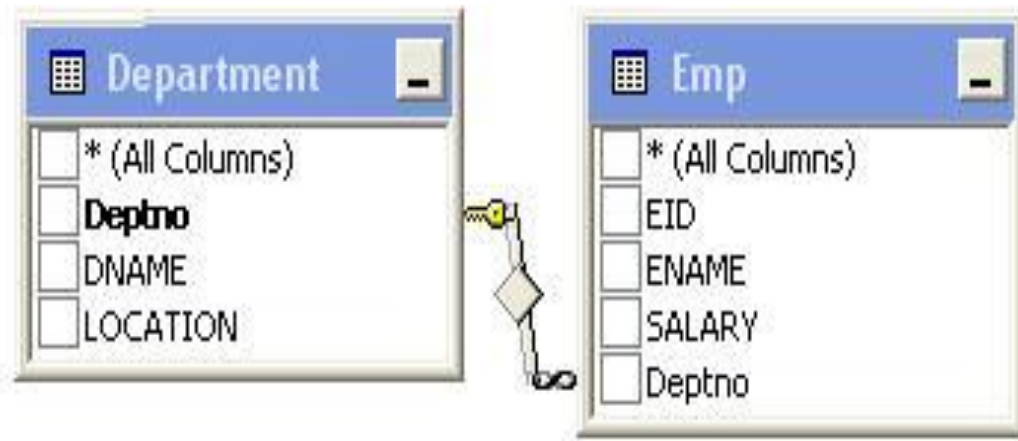
Insert Records Into Employee Table:

```
insert into Employee values(101,'Sai',35000,10)  
insert into Employee values(102,'Pavan',45000,20)  
insert into Employee values(103,'Kamal',74000,30)  
insert into Employee values(104,'Ravi',58000,40)
```

The below records are not allowed in to employee table:

```
insert into Employee values (105, 'Kamal', 74000, 50)
```

```
insert into Employee values (106, 'Ravi', 58000, 60)
```



When we impose the foreign key constraint and establish relation between the table, the following three rules will come into picture.

Rule1:- Cannot insert a value into the foreign key column provided that value is not existing under the reference key column of the parent table.

Rule2:- Cannot update the reference key value of a parent table provided that value has corresponding child record in the child table without addressing what to do with the child record.

Rule3:- Cannot delete a record from the parent table provided that records reference key value has child record in the child table without addressing what to do with the child record.

If we want to delete or update a record in the parent table when they have corresponding child records in the child table we are provided with a set of rules to perform delete and update operations known as cascade rules.

On delete cascade:- It is used to delete a key value in the parent table which is referenced by foreign key in other table all rows that contain those foreign keys in child table are also deleted.

On Update cascade:- It is used to Update a key value in the parent table which is referenced by foreign key in other table all rows that contains those foreign keys in child table are also updated.

If we apply this rule while creating the child table like below

create table **Emp**(EID int,ENAME varchar(50),SALARY money,Deptno int
foreign key references **Department**(Deptno)on delete cascade on update cascade)

Ex:-

- update **Department** set **Deptno**=222 where **Deptno**=20
- delete from **Department** where **Deptno**=222

Making a Relationship between Three Tables

CASE-1:

Create table CUSTOMER (CID Int primary key, CNAME Varchar (20), MAILID Varchar (40))

Insert customer values (1,'a','a@gmail.com'),
(2,'b','b@gmail.com'), (3,'c','c@gmail.com')

CASE-2

Create table PRODUCTS (PCODE Int primary key, PNAME varchar (50),
PRICE money)

Insert products values (10,'C', 500), (20,'C++', 1000), (30,'.NET', 35000),
(40,'SQL', 1800)

CASE-3

Create table ORDERS (ORID Int primary key, ORDATE date, QUANTITY
int,CID Int foreign key references CUSTOMER(cid) on update cascade on

delete cascade, PCODE Int foreign key references PRODUCTS(pcode) on update cascade on delete cascade)

Insert into ORDERS values (101,'2014/10/15', 3, 2, 10) //ALLOWED

Insert into ORDERS values (102,'2014/10/25', 3, 4, 50) //NOTALLOWED

Update CUSTOMER set cid=100 where cid=1

Delete from CUSTOMER where cid=100

Update PRODUCTS set orid=21 where orid=20

Delete from PRODUCTS where orid=30

Adding Constraint on an Existing Table:



Adding Primary Key on an existing table:

Ex: CREATE TABLE EMPLOYEE (EID INT, ENAME VARCHAR (50), SALARY MONEY)

Note: Before adding primary key constraint make the column is not null later add primary key like below.

Ex: ALTER TABLE EMPLOYEE ALTER COLUMN EID INT NOT NULL

Ex: ALTER TABLE EMPLOYEE ADD CONSTRAINT X PRIMARY KEY (EID) (Here 'x' is constraint variable)



Adding Unique Key on an existing table:

Ex: ALTER TABLE EMPLOYEE ADD CONSTRAINT X UNIQUE (ENAME)



Adding Check Key on an existing table:

Ex: ALTER TABLE EMPLOYEE ADD CONSTRAINT X CHECK (SALARY>8000)



Adding Foreign Key on an existing table:

Ex: ALTER TABLE DEPT ADD CONSTRAINT Y FOREIGN KEY (EID) REFERENCES EMPLOYEE (EID) ON UPDATE CASCADE ON DELETE CASCADE



Dropping Constraint from an existing table:

Ex: ALTER TABLE EMPLOYEE DROP CONSTRAINT X

JOINS IN SQL: Joins are used for retrieving the data from more than one table at a time. Joins can be classified into the following types.



EQUI JOIN



INNER JOIN



OUTER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN



FULL OUTER JOIN



NON EQUI JOIN



SELF JOIN



CROSS JOIN



NATURAL JOIN

EQUI JOIN: If two or more tables are combined using equality condition then we call as an Equi join.

Ex: WAQ to get the matching records from EMP and DEPT tables

Sol: SELECT * FROM EMP, DEPT WHERE (EMP.EID=DEPT.DNO) (NON-ANSI STANDARD)

Sol: SELECT E.EID, E.ENAME, E.SALARY, D.DNO, D.DNAME FROM EMP E, DEPT D WHERE E.EID=D.DNO (NON-ANSI STANDARD)

INNER JOIN: Inner join return only those records that match in both table

Ex: SELECT * FROM EMP E INNER JOIN DEPT D ON E.EID=D.DNO (ANSI)

OUTTER JOIN: It is an extension for the Equi join. In Equi join condition we will be getting the matching data from the tables only. So we loss UN matching data from the tables.

To overcome the above problem we use outer join which are used to getting matching data as well as UN matching data from the tables. This outer join again classified into three types

LEFT OUTER JOIN: It will retrieve or get matching data from both table as well as UN matching data from left hand side table

Ex: SELECT * FROM EMP LEFT OUTER JOIN DEPT ON
EMP.EID=DEPT.DNO;

RIGHT OUTER JOIN: It will retrieve or get matching data from both table as well as UN matching data from right hand side table

Ex: SELECT * FROM EMP RIGHT OUTER JOIN DEPT ON
EMP.EID=DEPT.DNO;

FULL OUTER JOIN: It will retrieve or get matching data from both table as well as UN matching data from left hand side table plus right hand side table also.

Ex: SELECT * FROM EMP FULL OUTER JOIN DEPT ON
EMP.EID=DEPT.DNO;

NON EQUI JOIN: If we join tables with any condition other than equality condition then we call as a non Equi join.

Ex: SELECT * FROM EMP, SALGRADE WHERE (SALARY > LOWSAL)
AND (SALARY < HIGHSAL)

SELF JOIN: Joining a table by itself is known as self join. When we have some relation between the columns within the same table then we use self join.

When we implement self join we should use alias names for a table and a table contains any no. of alias names.

Ex: SELECT E.EID, E.ENAME, E.SALARY, M.MID, M.ENAME
MANAGERSNAME, M.SALARY FROM EMP E, EMP M
WHERE E.EID=M.MID.

CROSS JOIN: Cross join is used to join more than two tables without any condition we call as a cross join. In cross join each row of the first table join with each row of the second table.

So, if the first table contain „m“ rows and second table contain „n“ rows then output will be „m*n“ rows.

Ex: SELECT * FROM EMP, DEPT

Ex: SELECT * FROM EMP CROSS JOIN DEPT

NATURAL JOIN: It is used to avoid duplicate column from the tables.

EX: SELECT EID, ENAME, SALARY, DNO, DNAME FROM EMP E, DEPT D
WHERE E.EID=D.DNO

Making To Joins Three Tables:

CREATE TABLE STUDENTS (SID Int, SNAME varchar (20), SMBNO char (10),
CID Int)

INSERT STUDENT VALUES (1,'aa','7894561233', 10), (2,'bb','9874563211',
20), (3,'cc','8749653215', 30), (4,'dd','7788996655', 40)

CREATE TABLE COURSES (CID int, CNAME Varchar (20), CFEE decimal (6, 2))

INSERT COURSES VALUES (10,'c', 500), (20,'c++', 1000), (50,'sql', 1800),
(60,'.net', 3500), (70,'sap', 8000)

CREATE TABLE REGISTER (SNO int, REGDATE date, CID Int)

INSERT REGISTER VALUES (100,'2014/10/20', 10), (101,'2014/10/21',
80), (102,'2014/10/22', 90)

Select * from course c inner join student s on c.cid=s.cid inner join register r on s.cid=r.cid

Select * from student s left outer join course c on s.cid=c.cid left outer join register r on c.cid=r.cid

TRANSACTION CONTROLL LANGUAGE

TRANSACTION: A transaction is a unit of work that is performed against a database or set of statement (Insert, Update and Delete) which should be executed as one unit.

- A transaction is the propagation of one or more changes to the database. For example, if you are inserting 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.
- The rule of transaction tells that either all the statements in the transaction should be execute successfully or none of those statement to be executed.

To manage transaction we have provide with transaction control language that provides a commands like

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK
- SAVE POINT

BEGIN TRANSACTION: Begin Transaction command is used to start the transaction. Begin Transaction with name is used to add nested transactions.

Syntax: Begin transaction
 <Write Statements>

COMMIT: Commit command is used to end the transaction and save the data permanent part of the database (or) it is used to make the transaction is permanent so we cannot undo or recall the records.

- Commit is used for saving the data that has been changed permanently because whenever you perform any DML (Data Manipulation Language) like UPDATE, INSERT OR DELETE then you are required to write Commit at the end of all or every DML operation in order to save it permanently.
- If you do not write Commit then your data will be restored into its previous condition.

Syntax: **Begin Transaction**
 <Write Statements>

Commit

Ex: BEGIN TRANSACTION

```
INSERT INTO EMPLOYEE VALUES(105,'KAMAL',62000,'MUMBAI')  
INSERT INTO EMPLOYEE VALUES(106,'SUJATHA',82000,'DELHI')  
  
COMMIT
```

- The above records are stored permanently into a table because we committed that records.so we cannot roll back in to its previous position.

ROLLBACK: Rollback command is used to undo the transactions and gets back to the initial state where transaction started.

- Whereas if you want to restore your data into its previous condition then you can write Rollback at any time after the DML queries has been written but remember once Commit has been written then you cannot rollback the data.
- Moreover you can only rollback the DML queries that have been written after the last commit statement. The concept of commit and rollback is designed for data consistency because many users manipulate data of the

same table, using the same database so the user must get updated data. That is why commit and rollback are used.

Syntax: **Begin Transaction**
 Rollback

Ex: BEGIN TRANSACTION
 DELETE FROM **EMPLOYEE** WHERE **EID**=105
 DELETE FROM **EMPLOYEE** WHERE **EID**=106

 BEGIN TRANSACTION
 ROLLBACK

- The above records we can rollback into a table because those records are not committed.

SAVEPOINT: Save point is used for dividing (or) breaking a transaction into multiple units. So that user will have a chance of roll backing a transaction up to a location.

- When a user sets a save point with in a transaction the save point defines a location to which a transaction can return if part of the transaction conditionally canceled.
- If a transaction is roll back to a save point, it must be proceed to completion of the transaction with commit statement or it must be cancelled altogether by rolling the transaction back to its beginning

Syntax: **Begin Transaction**

 Save transaction < transaction name>
 <Write Statements>

Ex: BEGIN TRANSACTION
 UPDATE **EMPLOYEE** SET **SALARY**=99000 WHERE **EID**=101
 UPDATE **EMPLOYEE** SET **SALARY**=88000 WHERE **EID**=102
 SAVE TRANSACTION **S1**
 UPDATE **EMPLOYEE** SET **SALARY**=77000 WHERE **EID**=103
 UPDATE **EMPLOYEE** SET **SALARY**=66000 WHERE **EID**=104
 SAVE TRANSACTION **S2**
 UPDATE **EMPLOYEE** SET **SALARY**=55000 WHERE **EID**=105
 UPDATE **EMPLOYEE** SET **SALARY**=44000 WHERE **EID**=106



In the above case we are dividing or breaking the transaction into three units. so we have a chance of rollbacking either completely i.e six statements get roll back (or) roll back save point S1 i.e four statements (103 to 106) (or) roll back save point S2 i.e two records (105, 106) only

CASE 1: BEGIN TRANSACTION
ROLLBACK



All records will roll back i.e complete records (six records)

CASE 2: BEGIN TRANSACTION
ROLLBACK TRANSACTION S1



We can roll back four records only i.e 103 to 106.

CASE 3: BEGIN TRANSACTION
ROLLBACK TRANSACTION S2



We can roll back two records only i.e 105 and 106

Sub Query: A select query contains another select query is called sub Query. In this, there will be two queries those are called as inner query and outer query. When it is executed, first inner query is executed later outer query will be executed

Syntax: select * from <Table Name> where (condition) (select * from..... (Select * from..... (select * from.....)));

Types of Sub Queries: We have two types of sub queries

Nested Sub queries: In a sub query, Outer query depends on result of inner query is called as Nested sub query.

Examples:

1) WAQ to find the details of employee who is earning the highest salary.

Sol: select * from **employee** where **Salary**=(select MAX(salary) from **employee**)//**subquery**

2) WAQ to find the details of employee who is earning second highest salary.

Sol: select * from **employee** where **Salary**=(select MAX(salary)from **employee** where **Salary**<(select MAX(salary) from **employee**)) //multiple sub query

3) WAQ to find the details of employee who is earning third highest salary.

Sol: select * from **employee** where **Salary**=(select MAX(salary)from **employee** where **Salary**<(select MAX(salary) from **employee** where **Salary** <(select MAX(salary) from **employee**))) //nested subquery

4) WAQ to display employee details who are working in .NET department.

Sol: select * from **employee** where EID IN (select EID FROM **employee** where DNAME='.NET')

5) WAQ to display employee details who are working in JAVA or HR department.

Sol: select * from **employee** where EID IN (select EID FROM **employee** where DNAME='.NET' OR DNAME='HR')

Co-Related Sub queries: In a sub query, inner query depends on result of outer query is called as Co-Related sub query.

Outer query will execute first and value of the outer query will be used by co-related sub query i.e. inner query.

When outer query is executed then the copy of the table will be stored in memory, later co related sub query will check the values and will give Ranks to the outer query rows in the memory ,according to the ranks the result will be displayed.

Note: - To find Top n salaries list use “n>”.To find nth highest salary use “n-1”.

Syntax: SELECT * FROM Employee Emp1 WHERE (N-1) = or N> (SELECT COUNT (DISTINCT (Emp2.Salary)) FROM Employee Emp2WHERE Emp2.Salary > Emp1.Salary)

Disadvantage: Sub query execution will be fast and co related sub query execution is slow because it will check every row of inner query with every row of the outer query.

Examples:

6) WAQ to display top 2 salaries list from employee table.

Sol: Select * from employee E where 2>(select count(salary) from employee M where M.salary>E.salary)

(Or)

Sol: Select * from employee E where 2>(select count(Distinct salary) from employee M where M.salary>E.salary)

Note: - Here Distinct Key will be used when the table contain duplicate values.

7) WAQ to display least 2 salaries list from employee table.

Sol: Select * from employee E where 2>(select count(salary) from employee M where M.salary<E.salary)

(Or)

Sol: Select * from employee E where 2>(select count(Distinct salary) from employee M where M.salary<E.salary)

8) WAQ to display “N th” salary from employee table.

Sol: Select * from employee E where 0=(select count(salary) from employee M where M.salary>E.salary)

Note: When condition is M.salary>E.salary / M.salary<E.salary then

N=0 then first highest salary / first least salary

N=1 then second highest salary/second least salary

N=2 then third highest salary / third least salary

9) WAQ to get the details of the department in which employee are working.

Select * from department D Exists (select * from employee E
where E.deptno=D.deptno)

10) WAQ to get the details of the department in which employee are
not working.

Select * from department D not Exists (select * from employee E
where E.deptno=D.deptno)

Syntax to Find Any Position Record From A table

SELECT * FROM(SELECT *,ROW_NUMBER() OVER (ORDER BY
empid) AS **RowNum** FROM **Employee**) **employee** WHERE **RowNum**
=15

Syntax to Delete Duplicate Records from a table contains more than two same duplicate values:

| SID | SNAME | FEE | rownum |
|-----|--------|-------|--------|
| 10 | Sai | 12000 | 1 |
| 20 | Siddhu | 45000 | 1 |
| 30 | Meena | 65000 | 1 |
| 30 | Meena | 65000 | 2 |
| 30 | Meena | 65000 | 3 |
| 30 | Meena | 65000 | 4 |
| 30 | Meena | 65000 | 5 |

with **duplicates** as

(select *,ROW_NUMBER() over(partition by sid,sname, **fee** order by
sid,sname,fee) **rownum** from **student**)

delete from **duplicates** where **rownum** > 1

select the complete above query and execute then we delete all duplicate records
which are greater than 1 i.e.output is like below

| SID | SNAME | FEE | rownum |
|-----|--------|-------|--------|
| 10 | Sai | 12000 | 1 |
| 20 | Siddhu | 45000 | 1 |
| 30 | Meena | 65000 | 1 |

INDEXES IN SQL:

Why We Need Indexes:

1. Generally a library has a huge collection of books, files, etc... A student requests the librarian for a book of Microsoft SQL Server 2008, if we think without an index the librarian had to find this without any help she/he has to search one by one! This must be time consuming; so with a proper arrangement, that is with the help of an index, it very much easier and faster to find out the desired one.
2. One of the most important routes to high performance in a SQL Server database is the index. Indexes speed up the querying process by providing quickly access to rows in the data tables, similarly to the way a book's index helps you find information quickly within that book.

What is INDEX:

- **Index** is a database object which is used for the quick retrieving of the data from the table.
- An index contains keys built from one or more columns in the table and map to the storage location of the specified data.
- By using indexes we can save time and can improve the performance of database queries and applications.
- When we create an indexes on any column, SQL server internally maintain a separate table called index table. So that when ever user trying to retrieve the data from existing table depends on index table SQL server directly go to the table and retrieve required data very quickly.

- In a table we can use max 250 indexes. The index type refers to the way the index is stored internally by SQL Server. So a table can contain the two types of indexes.
 1. Clustered
 2. Non-Clustered

Clustered Index:

- The only time the data rows in a table are stored in sorted (ascending order only) order structure is when the table contains a clustered index. When a table has a clustered index then is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure.
- A table can have only 1 clustered index on it, which will be created when primary key constraint is used in a table.

Non-Clustered Indexes:

- Non-clustered indexes will not have any arrangement order (Unordered structure) of the data in the table. In a table we can create 249 non-clustered indexes.
- If we don't mention clustered indexes in a table then default is stored as non-clustered indexes.

Syntax: Create Index <Index Name> on <Table Name> (Column Name);

EX: Create index demo index on EMP (Eid)

VIEWS IN SQL: View is database object which is like table but logical. We can call it as a logical or virtual table because it does not has a physical existence.

- It is a logical table use to get the required information from the table. View will be created by using select statement and table used for the creation of the view is called as base table.
- View will not store records in it and will not occupy memory space with help of structure existing in it and records will be displayed from table.

- View is logical representation or virtual representation .it is a dependent where as table an independent is because view is extracted from the table.
- If we want to access the data from the table it's not necessary to change the data direct to the table but we can access by having a view.
- Views are used for security purpose in databases, views restricts the user from viewing certain column and rows means by using view we can apply the restriction on accessing the particular rows and columns for specific user.
- Views display only those data which are mentioned in the query, so it shows only data which is returned by the query that is defined at the time of creation of the View.

Why We Need Views: To protect the data. If you have a table containing sensitive data in certain columns, you might wish to hide those columns from certain groups of users. For instance, customer names, addresses and their social security numbers might all be stored in the same table; however, for lower level employees like shipping clerks, you can create a view that only displays customer name and address. You can grant permissions to a view without allowing users to query the original tables.

- A view is a logical table but what it stores internally is a select statement that is used for creating the view. So that whenever a user performs any operation on the view like select, insert, update or delete internally the view performs those operations on a table.
- Simply we can say that view will act as an interface between the data provider (Table) and the User.

View is created based on a table any changes that are performed on the table reflects into the view any changes performed on the view reflect into the table also. View is classified into two types. These are

- Simple view(Updatable view)
- Complex view(Non-Updatable view)

Simple view: we create a view based on one table is called simple view or Updatable view.

Complex view: we create a view based on more than one table is called complex view or Non-Updatable view.

Syntax: create view <view name> as select * from <table name>

Ex1: create view **simpleview** as select * from **emp**;

Ex2: CREATE VIEW **COMPLEXVIEW** AS SELECT
E.EID,E.ENAME,D.DNO,D.DNAME FROM **EMP** E INNER JOIN DEPT
D ON E.DEPTNO=D.DEPTNO

T/SQL Programming

- T/SQL stands for “Transact Structure Query Language. It is an extension of SQL language. This T/SQL is same as PL/SQL in oracle.
- In SQL we can execute single line statement only where as in T/SQL we can execute block of statements at a time.
- SQL does not support conditional and looping statements like IF-Else and While loop. But we can implement these conditional and looping statements in T/SQL.
- SQL language will not provide reusability facilities where as T/SQL language will provide reusability facilities by defining objects such as Procedures and Functions.
- T/SQL commands can be embedded inside the programs where program is a block of code.
- T/SQL Program blocks can be divided into two types. Those are
 1. Anonymous Blocks
 2. Sub-Program Blocks

Anonymous Blocks: Anonymous Blocks are called as unnamed block of code which is executed at any point of time and does not store on database. These blocks can be written on a query window and execute.

Sub-Programs: Sub program Blocks are called as named block of code which is executed at any point of time and stored on database. These blocks are providing reusability of code.

Declaring Variables In T/SQL Program:

Syntax: Declare @ <var> [as] <data type > [size].....

Ex: declare @ eid int; decalare @ename varchar (50)....

While declaring variable, we should be prefixed with @ symbol.

Assigning Values to variables: Values can be assigned by using a SET statement.

Syntax: Set @ <var>=<value>

Ex: Set @Eid=101; Set @ename="SAI";

Printing Values of Variables: If we want to print the values we can use the PRINT statement.

Syntax: Print @ <var>

Ex: Print @Eid;

Structure of T/SQL Program:

Syntax: Declare @ <var1> [data type][size].....

Set @ <var>=<values>

<Statements>;

Print @<var>.....

Ex1: Write a T/SQL program to input two values and interchange the variable values.

```
declare @a int,@b int,@c int;
set @a=10;
set @b=20;
set @c=@a;
set @a=@b;
set @b=@c;
print @a;
print @b;
```

Ex2: Write a T/SQL program to input student id,name,marks and find the total marks of a student.


```

declare @stdno int,@stdname varchar(50),@m1 int,@m2 int,@m3 int,@tm int;
set @stdno=101;
set @stdname='SAI';
set @m1=75;
set @m2=85;
set @m3=65;
set @tm=@m1+@m2+@m3;
print @stdno
print @stdname;
print @tm

```

Ex3: Write a T/SQL programer to perform arithmetic operation. declare @a int,@b int,@c int,@d int,@e int,@f int;

```

set @a=10;
set @b=12;
set @c=@a+@b;
set @d=@a-@b;
set @e=@a*@b;
set @f=@a/@b;
print @c;
print @d;
print @e;
print @f;

```

Conditional Statements: It is a block of code, which executes based on a condition.

If-Else Statement: In if-else conditional control statement, statements in if block gets executed only when the condition is true and statements in else block gets executed only when the condition is false.

Syntax:

If (condition)

{

Statements

}

Else

```
{  
Statements  
}
```

Ex: Write T/SQL program to find big number from two variables. declare @a int, @b int;

```
set @a=30;  
set @b=20;  
if(@a>@b)  
print 'a is big'  
else if(@a=@b)  
print 'Both are equal'  
else  
print 'B is big';
```

Ex: To find positive and negative.

```
declare @a int;  
set @a=10;  
if(@a>0)  
print 'a is positive'  
else  
if(@a=0)  
print 'a is  
neutral' else  
print 'a is negative'
```

Ex: To find the number is even or odd.

```
declare @a int  
set @a=4  
if((@a%2)=0)  
print 'a is even';  
else  
print 'a is odd';
```

EX: If there are multiple statements being enclosed between each block then we can put them under Begin and End Statements.

```
DECLARE @WEEK INT

SET @WEEK=DATEPART(DW, GETDATE())

IF @WEEK=1

    PRINT 'SUNDAY'

ELSE IF @WEEK=2

    PRINT 'MONDAY'
ELSE IF @WEEK=3

    PRINT 'TUESDAY'

ELSE IF @WEEK=4

    PRINT 'WEDNESDAY'

ELSE IF @WEEK=5

    PRINT 'THURSDAY'

ELSE IF @WEEK=6

    PRINT 'FRIDAY'

ELSE IF @WEEK=7

    PRINT 'SATURDAY'
```

CASE FUNCTION: The case function what we have discussed under the System Functions can also be used here as following:

```
DECLARE @WEEK INT

SET @WEEK=DATEPART(DW, GETDATE())

SELECT CASE @WEEK

WHEN 1 THEN 'SUNDAY'
```

```

        WHEN 2 THEN 'MONDAY'
        WHEN 3 THEN 'TUESDAY'

        WHEN 4 THEN 'WEDNESDAY'

        WHEN 5 THEN 'THURSDAY'

        WHEN 6 THEN 'FRIDAY'

        WHEN 7 THEN 'SATURDAY'

END

```

-This can be written in the second style of the CASE Statement also that has been discussed in the SQL as following:

```

DECLARE @WEEK INT

SET @WEEK=DATEPART(DW, GETDATE())

SELECT CASE

        WHEN @WEEK=1 THEN 'SUNDAY'

        WHEN @WEEK=2 THEN 'MONDAY'
        WHEN @WEEK=3 THEN 'TUESDAY'

        WHEN @WEEK=4 THEN 'WEDNESDAY'

        WHEN @WEEK=5 THEN 'THURSDAY'

        WHEN @WEEK=6 THEN 'FRIDAY'

        WHEN @WEEK=7 THEN 'SATURDAY'

END

```

While Loop: Sets a condition for the repeated execution of an SQL statement or statement block. The statements are executed repeatedly as long as the specified condition is true. The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords.

```

WHILE Boolean expression

```

[BEGIN]

< sql_statement | statement_block

> [BREAK]

< sql_statement | statement_block

> [CONTINUE]

< sql_statement | statement_block >

[END]

-If there are multiple statements being enclosed then we can put them under Begin and End Statements.

BREAK: Causes an exit from the innermost WHILE loop. Any statements that appear after the END keyword, marking the end of the loop, are executed.

CONTINUE: Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

Program 1:

DECLARE @X INT

SET @X=0

WHILE @X<10

BEGIN

SET @X=@X+1

PRINT @X

END

Program 2:

```
DECLARE @X INT
SET @X=0
WHILE @X<10
BEGIN
    SET @X=@X+1
    IF @X=6 BREAK
    PRINT @X
END
```

-In this case the break statement brings the control out of the loop printing from 1 to 5.

Program 3:

```
DECLARE @X INT
SET @X=0
WHILE @X<10
BEGIN
    SET @X=@X+1
    IF @X=6 CONTINUE
    PRINT @X
END
```

-In this case the continue statement will skip the print statement when the value of x is 6 so prints from 1 to 5 and 7 to 10.

Comments in TSQL: Comments will be ignored will executing the program, they will increase the readability and aids understanding of the program.

- Single Line Comments (--)
- Multi Line Comments (/* */)

Assinging values from columns into variables: Till now we were assigning static values to the variables using the SET statement, but we can also assign values from a column into the variables as following:

```
SELECT @<var>=<col_name> [, .....n] FROM <table_name>
[CONDITIONS]
```

```
SELECT @ENAME=ENAME FROM EMP WHERE EMPNO=1001
```

-A simple TSQL program which takes the Empno and prints the Name and Salary.

```
DECLARE @EMPNO INT, @ENAME VARCHAR(50), @SAL MONEY
```

```
SET @EMPNO=1005
```

```
SELECT @ENAME=ENAME, @SAL=SAL FROM EMP WHERE
EMPNO=@EMPNO
```

```
PRINT @ENAME + ' EARNs ' + CAST(@SAL AS VARCHAR)
```

-A Program which takes the Empno and increments the Salary of the person on the following criteria:

If Job is President increment with 10%

If Job is Manager increment with 8%

If Job is Analyst increment with 6%

If Job is any thing other incrmnt with 5%

```

DECLARE @EMPNO INT, @JOB VARCHAR(50)

SET @EMPNO=1005

SELECT @JOB=JOB FROM EMP WHERE EMPNO=@EMPNO

IF @JOB='PRESIDENT'

    UPDATE EMP SET SAL = SAL + SAL * 0.1 WHERE
    EMPNO=@EMPNO

ELSE IF @JOB='MANAGER'

    UPDATE EMP SET SAL = SAL + SAL * 0.10 WHERE
    EMPNO=@EMPNO

ELSE IF @JOB='ANALYST'

    UPDATE EMP SET SAL = SAL + SAL * 0.06 WHERE
    EMPNO=@EMPNO

ELSE

    UPDATE EMP SET SAL = SAL + SAL * 0.05 WHERE
    EMPNO=@EMPNO

```

SUB PROGRAMS: A sub program is a named block of code that is directly saved on the server and it can be executed when and where it is required. We have two types of sub programs in SQL server.

- Stored Procedures/Procedure
- Stored Functions/Functions

Stored Procedures/Procedure: A stored procedure is a database object which contains precompiled queries. Stored Procedures are a block of code designed to perform a task whenever we called.

Why we need stored procedure: Whenever we want to execute a SQL query from an application the SQL query will be first parsed (i.e. compiled) for execution where the process of parsing is time consuming because parsing occurs each and every time we execute the query or statement.

To overcome the above problem we write SQL statements or query under stored procedure and execute, because a stored procedure is a pre compiled block of code without parsing the statements gets executed whenever the procedures are called which can increase the performance of an application.

Advantages of Stored Procedure:

- As there is no unnecessary compilation of queries, this will reduce burden on database.
- Application performance will be improved
- User will get quick response
- Code reusability facility

How to Create Stored Procedures/Procedure (Without parameter):

Syntax: Create Procedures <Procedures Name>

As

Begin

<Statements>

End

Once the Procedure is created it is physically saved on the server as a Database Object which can be called whenever we required to the user.

We can call the above procedure from anywhere and from any application that is developed using JAVA (or) .NET languages

How to Call a Stored Procedures/Procedure:

Syntax: Exec <Procedure name>

Examples on without parameters Procedures:

1) Write a simple procedure program (with out parameters) to print WELCOME statement on the query window.

```
create procedure  
Test1 as  
begin  
print 'WELCOME TO STOREDPROCEDURES'  
end
```

Passing Parameters to Procedures: If we want to pass parameters to procedures then we are using the below syntax.

Syntax: Create Procedures <Procedures Name>

(Passing parameters)

As

Begin

<Statements>

End

Examples on Parameter Procedures:

1) Write a program to add the two values with Parameters Procedure.

```
create procedure test2(@a int,@b int)  
as  
begin  
declare @c int  
set @c=@a+@b;  
print 'Addition of two variables are:-'+cast(@c as  
varchar); end
```

2) Write a program to perform arithmetic operations of two values with Parameters Procedure.

```
create procedure test3(@a int,@b int)
as
begin
declare @x int,@y int,@z int,@s int
set @x=@a+@b;
set @y=@a-@b;
set @z=@a*@b;
set @s=@a/@b;
print 'Add of two variables are:-'+cast(@x as varchar);
print 'Sub of two variables are:-'+cast(@y as varchar);
print 'Mul of two variables are:-'+cast(@z as varchar);
print 'Div of two variables are:-'+cast(@s as varchar);
end
```

3)create a procedure to display employee details to the user

```
create procedure spselect
as
begin
select * from Employee
end
```

Output: execute spselect

4)create a procedure to accept employee ID and delete the record from employee table.

```
create procedure spdel
@eid int
as
begin
delete from Employee where EmpID=@eid
```

end

Output: exec spdel 4

5)create a procedure to accept employee ID and update the employee details from employee table.

```
create procedure spupdate
```

```
@eid int,@ename varchar(max),@salary money,@address char(30)
```

```
as
```

```
begin
```

```
update Employee set
```

```
EmpName=@ename,Salary=@salary,Address=@address where
```

```
EmpID=@eid
```

```
end
```

Output: exec spupdate 1,'kamal',88000,'vizag'

6)create a procedure to add records in employee table.

```
create procedure spinst
```

```
@eid int,@ename varchar(50),@salary money,@address varchar(50)
```

```
as
```

```
begin
```

```
insert into Employee
```

```
values(@eid,@ename,@salary,@address) end
```

Output: exec spinst 6 ,'Suman' ,41000 ,'chennai'

7)create a procedure to insert records in two tables.

```
create procedure spinserttwotables
@eid int,@ename varchar(50),@salary money,@Address
char(40),@Deptno int,@Dname char(30),@Loc char(20)
as
begin
insert into Employee values(@eid,@ename,@salary,@Address,@Deptno)
insert into Dept values(@Deptno,@Dname,@Loc)
end
```

Output: exec **spinsert** 7,'mohan',62000,'mumbai',10,'dotnet','hyd'

8) A Procedure with Default Values:

```
CREATE PROCEDURE PROC3(@X INT = 100, @Y INT)
AS
BEGIN
    DECLARE @Z INT
    SET @Z=@X+@Y
    PRINT 'The SUM of the 2 Numbers is: ' + CAST(@Z
AS VARCHAR)
END
```

-Executing the above procedure:

1. EXEC PROC3 200, 25
2. EXEC PROC3 @X=200, @Y=25
3. EXEC PROC3 @X=DEFAULT, @Y=25
4. EXEC PROC3 @Y=25

-In the 3rd and 4th case it uses the default value of 100 to the varibale X which has been given while creating the procedure.

9) A Procedure which takes the Empno and prints the Net Salary of the

Employee. CREATE PROCEDURE Net_Sal(@Empno int)

As

Begin

Declare @VSal money, @NSal money, @VPF money, @VPT money

EXEC Deductions @Empno, @VPF OUTPUT, @VPT OUTPUT

SELECT @Sal=Sal FROM Emp WHERE Empno=@Empno

SET @NSal = @VSal - @VPF - @VPT

Print „Net Salary of the Employee is: „, + Cast(@NSal as Varchar)

End

-Executing the above Procedure:

EXEC Net_Sal 1005

How To Drop Stored Procedure:

- Drop Procedure <Procedure Name>

Ex: Drop Procedure SP1

Stored Functions/Functions: Function is a block of code similar to a stored procedure which is also used to perform an action and returns result as a value. Function can be divided into two types, these are

1)Scalar-Valued Fuction: In this case we can return a attribute datatype as an output from the function.

Syntax: Create Function <Function Name> (@parameter <Data Type> [size])

Returns <return attribute data type>

As

Begin

<Function Body>

Return <return attribute name>

End

How to Call Scalar valued Functions:

Syntax: Select dbo.<Function Name> (value)

- 1) Create a function to return the cube of the given value.

create function **fcube** (@x int)

returns

int as

begin

return @x*@x*@x

end

Output:select **dbo.fcube**(3)

- 2) Create a function that takes an employee id and returns the salary of that employee.

```
create function fsal(@eid int)
returns money
as
begin
declare @sal money
select @sal=salary from employee where empid=@eid
return @sal
end
```

Output:select **dbo**.fsal(1)

2)Table-Valued Fuction:In this case we can return a table as an output from the function.

Syntax:

Create Function <Function Name> (@parameter <Data Type> [size])

Returns <Table>

As

Return <return select statement>

How to Call a Table-Valued Function:

Syntax: select * from **functionname**(value)

Ex: Create a function that accept the Address and returns the list of employee working in given address from the table.

```
create function ft1(@add
varchar(50)) returns table
as
return(select * from employee where address=@add)
```

Output:select * from **ft1**('hyd')

Ex: Create a function to get the deptno and return list of employee working in EMP and DEPT tables.

```
create function saidata(@deptno  
int) returns table  
as
```

```
return(select e.eid,e.ename,e.salary,d.deptno,d.dname,d.location  
from emp e inner join dept d on e.deptno=d.deptno where  
e.deptno=@deptno)
```

Output: Select * from saidata(10)

How To Drop Functions:

- Drop Function <Function Name>

Ex: Drop Function Saidata

Difference between Function And Procedure:

- A function must return a value where as procedure never returns a value.
- A procedure can have parameters of both input (with parameters) and output (without parameters) where as a function can have only input (with parameters) parameters only.
- In procedure we can perform select, insert, update and delete operation where as function can used only to perform select. Cannot be used to perform insert, update and delete operations.
- A procedure provides the option for to perform transaction management where as these operations are not permitted in a function.
- We call a procedure using execute command where as function are called by using select command only.

TRIGGERS: A trigger is a special type of procedure that will be used to provide restriction on the tables when a language event is executed. SQL server includes two types of triggers are

- DML Triggers
- DDL Triggers

DML Triggers: DML triggers execute when the user tries to modify or change data through data manipulation language events. Those are Inserting, Update and Delete statements on the table.

DML triggers can be used to enforce business rules and data integrity. With the help of a DML trigger we can enforce integrity which cannot be done with constraints.

Syntax: Create Trigger <Trigger Name> on <table Name>

For [Insert, Update, Delete]

AS

Begin

<Statements>

End

Ex: A trigger that will convert the name and location into upper case when the user inserts in lowercase.

```
create trigger per_trg
on person after insert
as
begin
declare @pid int, @pname varchar(50), @loc varchar(50)
select @pid=pid, @pname=pname, @loc=loc from inserted
update person set pname=upper(@pname), loc=upper(@loc) where pid=@pid
end
```

Ex: Create a trigger to restrict DML operations on the table

```
create trigger nnn on person
for insert,update,delete
as
begin
print 'DML Operation are Not
Allowed' rollback transaction
end
```

EX: A Trigger that will restrict the operations to be performed before 9 A.M and after 5 P.M

```
CREATE TRIGGER EMP_TRG
ON EMP AFTER INSERT, UPDATE, DELETE
AS
BEGIN
DECLARE @DT INT
SET @DT=DATENAME(HH, GETDATE())
IF @DT NOT BETWEEN 9 AND 16
BEGIN
ROLLBACK
RAISERROR('CANNOT PERFORM DML OPERATIONS NOW', 15, 1)
END
END
```

-After the trigger is created try to perform any DML Operations on the EMP table before 9 A.M and after 5 P.M the Trigger will fire and restrict the operations.

EX: A program which will restrict the Delete operation if the Job of the person is Manager.

```
ALTER TRIGGER EMP_DELETE_TRG
ON EMP AFTER DELETE
AS
BEGIN
DECLARE @JOB VARCHAR(50)
SELECT @JOB=JOB FROM
DELETED IF @JOB='MANAGER'

BEGIN
ROLLBACK

RAISERROR('CANNOT DELETE MANAGER FROM THE TABLE', 15,
1)

END

END
```

-To test the following Trigger execute the following statement:

```
DELETE FROM EMP WHERE EMPNO=1002
```

EX: A Trigger which will restrict to update the Salary of the Employee if the New Salary is less than the Old Salary.

```
CREATE TRIGGER EMP_UPDATE_TRG
ON EMP AFTER UPDATE
AS
BEGIN
DECLARE @OLDSAL MONEY
```

```
DECLARE @NEWSAL MONEY

SELECT @OLDSAL=SAL FROM DELETED

SELECT @NEWSAL=SAL FROM INSERTED

IF @OLDSAL > @NEWSAL

BEGIN

ROLLBACK

RAISERROR('NEW SAL CANNOT BE LESS THAN OLD SAL', 15, 1)

END

END
```

Dropping DML Triggers:

Syntax: Drop <Trigger> <Trigger Name>

EX: Drop Trigger rest drop

DDL Triggers: DDL triggers fire in response to a data definition language event like create, Alter, drop etc. A DDL trigger is a special type of procedure that executes in response to a server scoped or database scoped events.

Syntax:

Create Trigger <Trigger Name> on database after <Event type>

As

Begin

<Statements>

End

Ex: Write a trigger which restricts dropping of a table from the database.

```
create trigger restdrop on database after  
drop_table as  
begin  
rollback  
raiserror('Can not drop table under this  
database',1,1) end
```

Ex2: Write a trigger which restricts Creating of a table from the database.

```
create trigger restcret on database after create_table  
as  
begin  
rollback  
raiserror('Can not create table under this  
database',1,1) end
```

Ex3: Write a trigger which restricts Alter of a table from the database.

```
create trigger restalt on database after  
Alter_table as  
begin  
rollback  
raiserror('Can not Alter table under this database',1,1);end
```

Dropping DDL Triggers:

Syntax: Drop <Trigger> <Trigger Name> on Database

EX: Drop Trigger rest drop on database

Magic Tables: SQL Server allows you to define a Magic Table. Magic Tables are invisible tables or virtual tables. You can see them only with the help **Triggers** in SQL Server.

- Magic Tables are those tables which allow you to hold inserted, deleted and updated values during insert delete and update DML operations on a table in SQL Server.
- Basically there are two types of magic table in SQL server namely inserted and deleted magic tables update can be performed with help of these twos.

Generally we cannot see these two table, we can only see it with the help Trigger's in SQL server.

Inserted Magic Table: Whenever you insert a record on that table, that record will be shown in the **INSERTED** Magic Table. Now creating a trigger to see the data in Inserted Magic table.

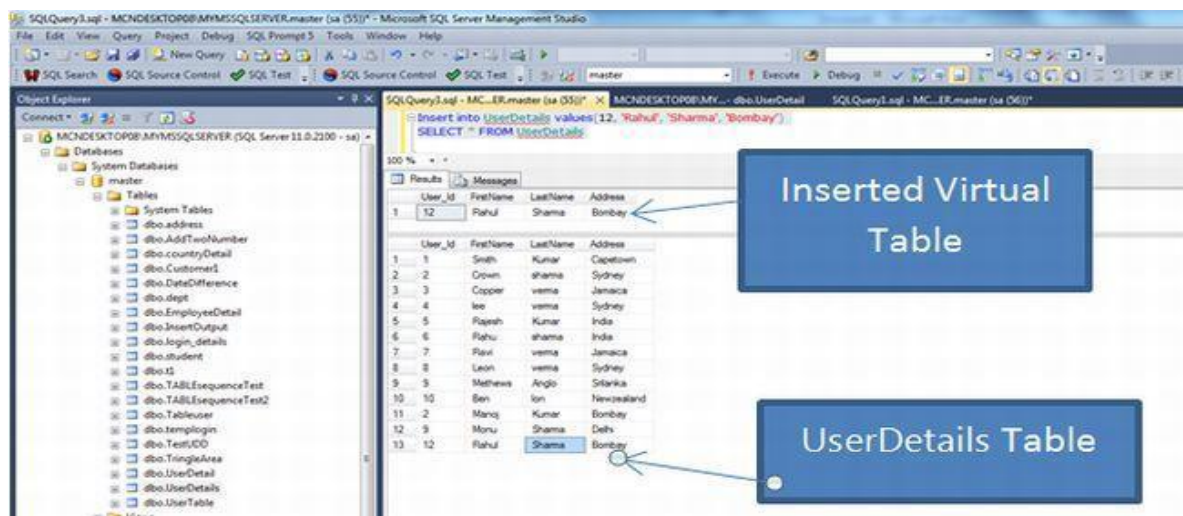
Example:

```
Create TRIGGER Trigger_ForInsertmagic  
ON Employee  
FOR INSERT  
AS  
Begin  
SELECT * FROM INSERTED  
End
```

Now insert a new record in Employee table to see data within Inserted virtual tables.

Insert into Employee **values** (12, 'Rahul', 25000, "HYD")
SELECT * FROM Employee

Now press **F5** to execute it.



Deleted Magic Table: Whenever you delete the record on that table, that record will be shown in the **DELETED** Magic Table Only. To create a trigger to see the data in the deleted Magic table use the following,

Example:

```
Create TRIGGER Trigger_Fordeletemagic
ON Employee
FOR DELETE
AS
Begin
SELECT * FROM Deleted
End
```

Now delete a record in the Employee table to see the data in the Deleted virtual tables.

```
Delete from Employee where Eid=12
SELECT * FROM Employee
```

Update the Record in Table: To update the record in the Employee table, we use it for both virtual/Magic tables. One shows the inserted table and the other shows the deleted table. The following trigger defines both the inserted table and the deleted table:

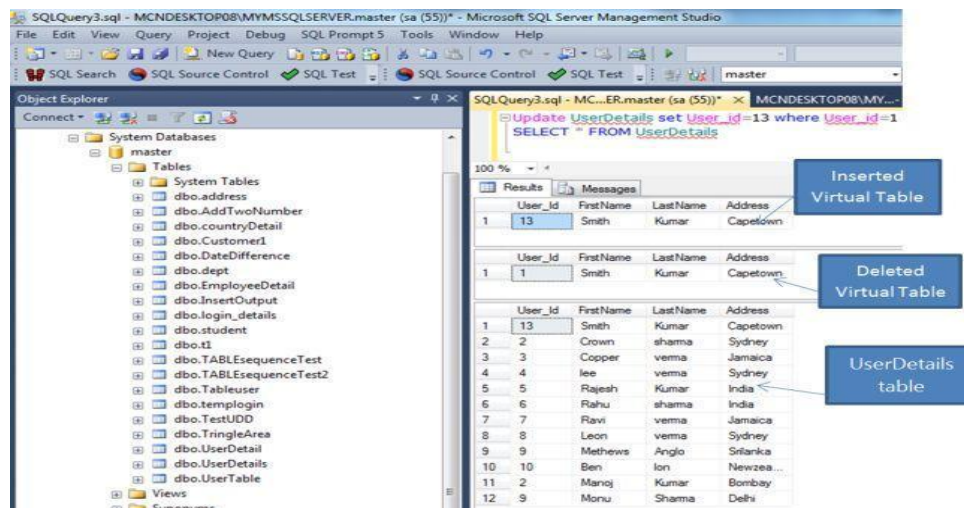
Example:

```
Create TRIGGER Trigger_ForInsertdeletemagic
ON Employee
FOR UPDATE
AS
Begin
SELECT * FROM INSERTED
SELECT * FROM DELETED
End
```

Now update the records in the Employee table to see the data in the inserted and deleted virtual tables.

```
Update employee set ename="sai" where Eid=12
SELECT * FROM Employee
```


Now press F5 to execute it.



Exception Handling: We handle errors of a program both in a programming language as well as databases also. whereas handling an error in a programming language needs stopping the abnormal termination and allowing the statements which are not related with the error to execute where as handling as error in sqlserver means stopping the execution of statements which are related with the error

Handling Errors In SQL Server: From sqlserver 2005 we are provided with a structure error handling mechanism with the help of TRY and CATCH blocks which should be used as following,

Begin Try

<Statements>

End Try

Begin Catch

<Statements>

End Catch

Ex: A procedure for dividing two numbers

```
CREATE PROCEDURE PDIV(@X INT,@Y INT)
AS
BEGIN
DECLARE @Z INT
BEGIN TRY
SET @Z=@X/@Y
PRINT 'THE RESULT IS:-' +CAST(@Z AS CHAR)
END TRY
BEGIN CATCH
PRINT ERROR_Message()
END CATCH
END
```

Exec PDIV 100,5

Error Message(): It is used to display the information about the error occurred.

CURSOR:Cursor is a memory location for storing database tables. cursor is a temporary work area allotted to the client at server when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

This temporary work area is used to store the data retrieved from the database, and manipulate this data. 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 the *Result set*.

There are two types of cursors in T/SQL:

Implicit Cursors: These cursors will be created by SQL server by default when select statement will executed. Select statement will show records in the table as a set or result set.

Explicit Cursors: When user can create a memory location to store the tables then it is called as Explicit Cursors. These cursors will access the records in the table record by record or one by one only. Whenever we want to go for record by record manipulation then explicit cursors will be used.

Steps To Create Cursor:

1)Declaring A Cursor:In this process we define a cursor.

Syntax: Declare <cursorname> cursor for < select statement>

2)Opening A 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 < cursorname>

3)Fetching Data From The Cursor:In this process we access row by row from cursor.

Syntax: Fetch first/last/next/prior/absolute n/relative n from <cursorname> into <variables>

4)Closing A Cursor: In this Process,it releases the current result set of the cursor leaving the datastructure available for reopening.

Syntax: Close <cursorname>

5) Deallocate A Cursor: It removes the cursor reference and deallocate it by destroy the data structure.

Syntax: Deallocate <cursorname>

@@Fetch Status: It is global variable use to check wheather cursor variable contains records ornot.if record is there then the value will be zero other wise value will be -1.

Example To Work with Cursor:

Ex: Create an explicit cursor to display all the records from the table.

Sol: declare @**dno** int,@**dname** char(20),@**loc** varchar(20)
declare **c1** cursor for select * from **dept**
open **c1**
fetch next from **c1** into @**dno**,@**dname**,@**loc**

```

while @@FETCH_STATUS=0
begin
print @dno
print @dname
print @loc
fetch next from c1 into @dno,@dname,@loc
end
close c1
deallocate c1

```

Ex: Create an explicit cursor to display salaries of each employee in the table.

```

declare @ename varchar(50),@sal money
declare empcur cursor for select name,sal from employee
open empcur
fetch next from empcur into @ename,@sal
while @@FETCH_STATUS=0
begin
print 'Salary Of'+' '+@ename+'is:-'+cast(@sal as varchar)
fetch next from empcur into @ename,@sal
end
close empcur
deallocate empcur

```

Ex: Write a program to increment the salaries of all the employee basing on the following criteria President 10%,Manager 5% and others 3%.

```

Declare @eno int,@job varchar(20)
Delcare empcur cursor for select Eid,Job from
employee Open empcur
Fetch next from empcur into
@eno,@job While @@Fetch_Status=0
Begin
If @job="president"
Update employee set sal+=sal*0.10 where eid=@eno

```

```

Else if @job="Manager"
Update employee set sal+=sal*0.05 where eid=@eno
Else
Update employee set sal+=sal*0.03 where eid=@eno
Fetch next from empcur into @eno,@job
End
Close empcur
Deallocate empcur

```

Forward only and Scroll Cursors:

If a cursor is declare as forward only it allows you to navigate only to the next records in sequential order and more over it supports only a singleton fashion method that is fetch next(one-by-one) where as a scroll cursor allows you to navigate/fetch Bidirectionally that is top- bottam or bottom-top also.And it supports six different fetch methods are
Fetch Next,Fetch First,Fetch Last,Fetch Prior,Fetch Absolute ,Fetch Relative.

Ex: Create an explicit cursor to fetch the records One-by-One manner(First-Last) from the table.

```

Sol: declare c1 cursor for select * from dept
      open c1
      fetch next from c1
      while @@FETCH_STATUS=0
      begin
      fetch next from c1
      end
      close c1
      deallocate c1

```

Ex: Create an explicit cursor to fetch the records from bottom -first (Last-First)from the table.

```

Sol: declare c1 cursor scroll for select * from dept
      open c1
      fetch last from c1

```

```
while @@FETCH_STATUS=0
begin
    fetch prior from c1
end
close c1
deallocate c1
```

Ex: Create an explicit cursor on fetching methods.

```
declare @id int
declare e cursor scroll
for select sid from
student open e
fetch next from e into @id
print @id
fetch last from e into
@id print @id
fetch prior from e into
@id print @id
fetch absolute 3 from e into
@id print @id
fetch relative -1 from e into
@id print @id
fetch first from e into
@id print @id
close e
deallocate e
```

Static & Dynamic Cursors: If a cursor is declare as static after opening the cursor any modifications that are performed to the data in the table will not be reflected into cursor so the cursor contains old values only in it.

```
Declare @sal money
Declare c1 cursor static for select sal from employee
Where eid=100
Open c1
```

Update employee set sal=25000 where eid=100

Fetch next from c1 into @sal

Print @sal

Close c1

Deal locate c1

Before executing the above program verify the salary of employee 100 and then execute the program even if the program is updating the salary in the table the fetch statement will still display us the old value of the table only but not the new value.

If we want the change made on the table to be reflected into the cursor after opening the cursor declare the cursor as dynamic

DATA CONTROL LANGUAGE

Authentication: Authentication is a process of verifying the credentials of a user to login into the system.

Authorization: Authorization is process of verifying whether the user has permissions to perform any operation on the database.

Data Control Language: DCL commands are used to enforce database security in multiple users' database environment. These are two types....

- GRANT
- REVOKE

GRANT: Grant command is used for giving a privilege or permission for a user to perform operations on the database.

Syntax: GRANT <Privilege Name> on <object name>

To {User} [With GRANT OPTION]

Privilege Name: Used to granted permission to the users for some rights are ALL, EXECUTE and SELECT.

Object Name: It is the name of database objects like Table, Views and Stored Procedure etc....

User: Used for to whom an access rights is being granted.

With Grant Option: Allows a user to grant access rights to other users.

REVOKE: Revoke command removes user access rights / privileges to the database OR taking back the permission that is given to a user.

Syntax: Revoke <privilege name> on <object name > from {user}

Normalization: Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process are,

- Eliminating redundant data (for example, storing the same data in more than one table) and
- Ensuring data dependencies make sense (only storing related data in a table).

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

There are several benefits for using Normalization in Database.

Benefits:

- a. Eliminate data redundancy
- b. Improve performance
- c. Query optimization
- d. Faster update due to less number of columns in one table
- e. Index improvement

There are different types of Normalizations form available in the Database. Let's see one by one.

1. First Normal Form (1NF): First normal form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table.

- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).
 - a. Remove repetitive groups
 - b. Create Primary Key

Example:

| Name | State | Country | Phone1 | Phone2 | Phone3 |
|-----------|-------|---------|-----------------------|--------------|--------------|
| John | 101 | 1 | 488-511-3258 | 781-896-9897 | 425-983-9812 |
| Bob | 102 | 1 | 861-856-6987 | | |
| Rob | 201 | 2 | 587-963-8425 | 425-698-9684 | |
| PK | | | [Phone No's] | | |
| ? | | | ? | | |
| ID | Name | State | Country | Phone | |
| 1 | John | 101 | 1 | 488-511-3258 | |
| 2 | John | 101 | 1 | 781-896-9897 | |
| 3 | John | 101 | 1 | 425-983-9812 | |
| 4 | Bob | 102 | 1 | 861-856-6987 | |
| 5 | Rob | 201 | 2 | 587-963-8425 | |
| 6 | Rob | 201 | 2 | 425-698-9684 | |

2. Second Normal Form (2NF): Second normal form (2NF) further addresses the concept of removing duplicative data:

- Meet all the requirements of the first normal form.
- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.

Remove columns which create duplicate data in a table and related a new table with Primary Key - Foreign Key relationship

| ID | Name | State | Country | Phone |
|----|------|-------|---------|--------------|
| 1 | John | 101 | 1 | 488-511-3258 |
| 2 | John | 101 | 1 | 781-896-9897 |
| 3 | John | 101 | 1 | 425-983-9812 |
| 4 | Bob | 102 | 1 | 861-856-6987 |
| 5 | Rob | 201 | 2 | 587-963-8425 |
| 6 | Rob | 201 | 2 | 425-698-9684 |

| ID | Name | State | Country |
|----|------|-------|---------|
| 1 | John | 101 | |
| 2 | Bob | 102 | |
| 3 | Rob | 201 | |

| PhoneID | ID | Phone |
|---------|----|--------------|
| 1 | 1 | 488-511-3258 |
| 2 | 1 | 781-896-9897 |
| 3 | 1 | 425-983-9812 |
| 4 | 2 | 587-963-8425 |
| 5 | 3 | 587-963-8425 |
| 6 | 3 | 425-698-9684 |

3. Third Normal Form (3NF): Third normal form (3NF) goes one large step further:

- Meet all the requirements of the second normal form.
- Remove columns those are not dependent upon the primary key.

Country can be derived from State also... so removing country

| ID | Name | State | Country |
|----|------|-------|---------|
| 1 | John | 101 | 1 |
| 2 | Bob | 102 | 1 |
| 3 | Rob | 201 | 2 |

4. Fourth Normal Form (4NF): Finally, fourth normal form (4NF) has one additional requirement:

- Meet all the requirements of the third normal form.
- A relation is in 4NF if it has no multi-valued dependencies.

If PK is composed of multiple columns then all non-key attributes should be derived from FULL PK only. If some non-key attribute can be derived from partial PK then remove it. The 4NF also known as BCNF NF

| ID | Name | State |
|----|------|-------|
| 1 | John | 101 |
| 2 | Bob | 102 |
| 3 | Rob | 201 |

5. Fifth Normal Form (5NF): A database table is said to be in 5NF if it is in 4NF and contains no redundant values or We can also said a table to be in 5NF if it is in 4NF and contains no join dependencies.

