**SSMS (SQL Server Management Studio)**

**What is database?**

Organized collection of data stored in an electronic format. So, database is an electronic system it allows to access, Manipulate and update data.

**What is DBMS?**

DMBMS is system software for creating and managing databases.

Different type database:

* Oracle (is also known as PL SQL): (procedural language extension to Structured Query Language)
* MSSQL(Open source SQL by microsoft)
* MYSQL(developed by Microsoft) : Also known as T SQL(Transcat SQL)

**What is SQL**

SQL Stands for Structured Query Language which is a standard language for accessing and manipulating database. (American National Standards Institute) ANSI introduced SQL.

**Types of Database Architecture:**

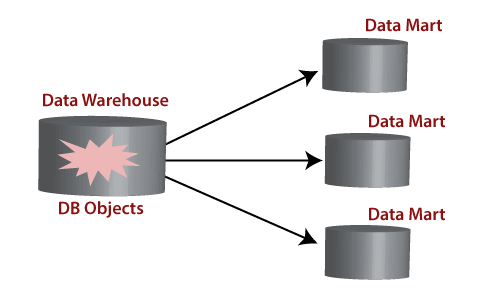
* File-Server
* Client –server

**What is data warehouse**

A data warehouse, also known as an enterprise data warehouse, is a system used for reporting and data analysis and is considered a core component of business intelligence. Data warehouses are central repositories of integrated(compact) data from one or more disparate(different) sources,

**What is data mart**

A data mart is a subset of a data warehouse and it focused on a particular line of business, department, or subject area.

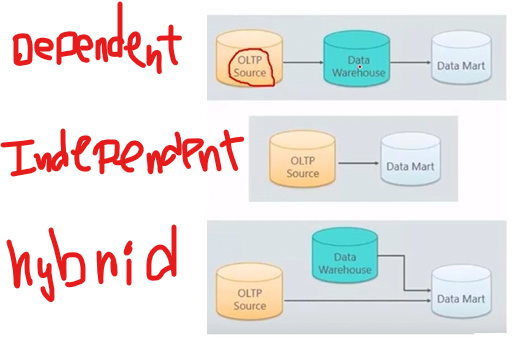


|  |  |
| --- | --- |
| **Data warehouse** | **Data Mart** |
| Enterprise-wide data | Department wide data |
| Multiple subject area | Single subject area |
| Multiple data sources | Limited Data sources |
| Occupies large memory | Occupies limited memory |
| Longer time to implement | Short time to implement |

**Types of Datamart:**

Three basic types of data marts are :

* Dependent
* Independent
* Hybrid.

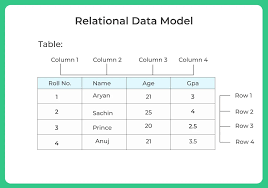
****

**Data model:**

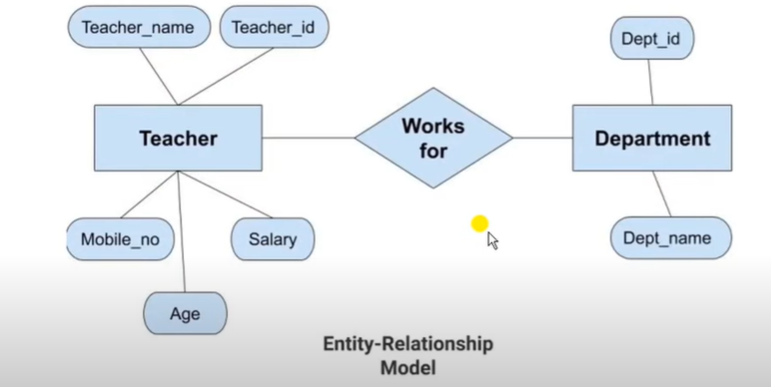
* A data model in DBMS, is the concept of tools that are developed to summarize the description of the database.
* It defines how the logical structure of a database is modeled.
* It defines how data is connected to each other and how they are processed and stored inside the system.
* It describes the design of a database at each level of data abstraction.

**Types of Data Model:**

1. **Relational Data Model:** This type of model designs the data in the form of rows and columns within a table.

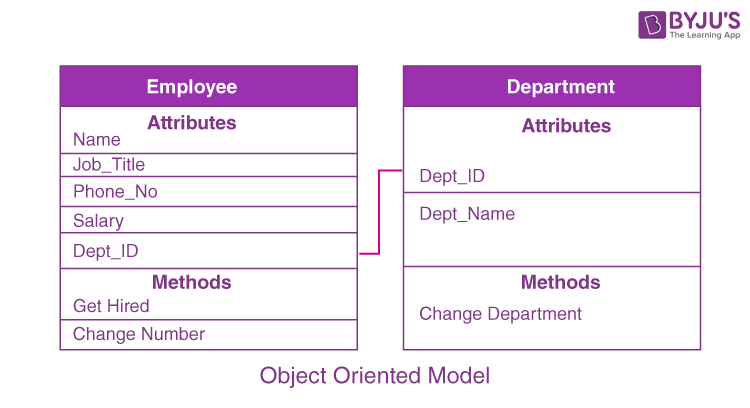
****

1. **Entity-Relationship Data Model:** An ER model is the logical representation of data as objects and relationships among them.

****

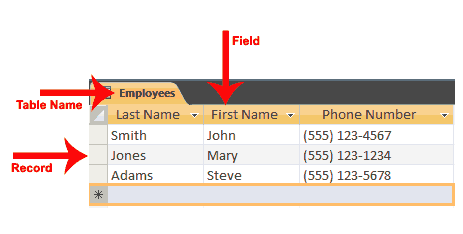
* The entities are teacher and department.
* The attributes of teacher entity are mobile no, teacher name etc.
* The attributes of department entity are dept\_id, dept\_name.
* Two entities are connected using the relationship. Here teacher works for students.

1. **Object-based Data Model**



**Tables in SQL**

* A table is a database object with comprises of rows and columns.
* In SQL Rows are known as Record. Each individual entry of a table is called the record. Record also provided complete information of the entities.
* Column is known as Field. A filed provides specific information about data in the table.

****

**Database manipulation:**

|  |
| --- |
| **CREATE: Create database ert** |
| **USE: Use ert** |
| **DROP: Drop database ert** |

DATA Type of SQL:

In MySQL there are three main data types:

* string
* numeric
* date and time

Numeric Data Type:

| **Type** | **Storage (Bytes)** | **Minimum Value Signed** | **Maximum Value Signed** |
| --- | --- | --- | --- |
| TINYINT | 1 | 0 | 255 |
| SMALLINT | 2 | -32768 | 32767 |
| INT | 4 | -2147483648 | 2147483647 |
| BIGINT | 8 | -263 | 263-1 |
| Decimal(a,b) |  | b=after decimal | a=total digit=before decimal+ after decimal=total length |

Why we are using data type?

The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

\*\*Why we are putting range in a data type?

Suppose if we are using tinyint data type to create age column, so tinyint range is 0-255 and it takes1 bytes (storage).

If we are taking bigint data type replace of tinyint data type then we can see bigint range is  0 to 18446744073709551615 and it takes 8 bytes (storage).After that we put 100 data under this column then space will be take 800bytes( storage)..

If we are using tinyint then it will take 100 bytes( storage).

So that’s why we are using range in data type in SQL.

Character data Type

**1 character =1 byte**

|  |  |  |
| --- | --- | --- |
| **DatabType** | **Range** | **Description** |
| **char** | 8,000 characters. | If we are putting 10 range between char and we are not passing 10 characters then sql will store blank spaces with character. That’s why we are not using char. |
| **Varchar (variable length character)** | 8,000 characters. | For above reason we are using varchar. |
| **Nvarchar(variable length character)** |  | NATIONAL CHARACTER VARYING. When we are using different language character data then use navarchar. before character we always put N. For non-English data type we are using n varchar. |
| **text=Varchar(max) /nvacar(max)** |  | Maximum character we can define for above mentioned data type is 8000.If we want to store more than 8000 character then we can use VARCHAR (MAX)/NVBARCHAR (MAX). It holds 2GB data. |

* text= Varchar(max) is replacement of text.

Char (10) ---10byte

Varchar (10) ---10byte

Nvarchar(10)---20 byte

DATE AND Time data type

|  |  |  |
| --- | --- | --- |
| **TYPE** | **FORMAT** | **Description** |
| **Time** | hh:mm:ss[.nnnnnnn] | Only time will return |
| **Date** | YYYY-MM-DD | Date only returns |
| **DATETIME** | YYYY-MM-DD hh:mm:ss[.nnn] | Date and time bot will return |

Constrain

SQL constraints are used to specify rules for the data in a table. Constrain are used after the data type. If table is already created then we can not add constrain through the alter command.

**Types of SQL Constraints**

* **NOT NULL** Constraint.

If we apply **not null** constrains on any column then, that column should not be allow any null value.

* **UNIQUE** Constraint.

If we apply unique constrains any column so, that column will not allow any duplicate values

* **DEFAULT ()** Constraint.

The DEFAULT constraint is used to set a default value for a column. The default value will be added to all new records, if no other value is specified.

Eg- default('india')

* **PRIMARY KEY** Constraint.

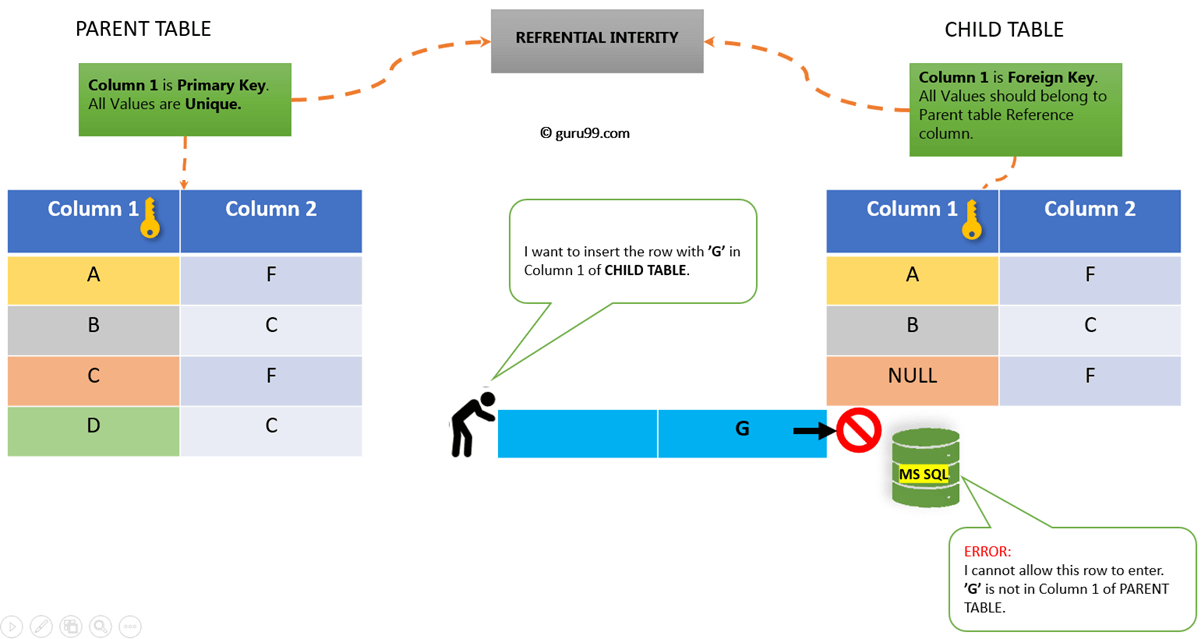
It is a combination of not null and unique constrain. It means primary key not allows any null and duplicate values. It is a key which can identify each and every record. One major difference is there between primary key and other constrain that is we can use multiple constrain on a table but we can use only one primary key in a table.

If create any column with primary key and that key will sort data in that particular column.

* **FOREIGN KEY** Constraint.

It is an attribute or set of attributes that references to primary key of same table or another table.

* Column name of primary key on parent table and column name of foreign key on child table will not require too same.
* In one table have multiple foreign keys in column.
* It maintains referential integrity.
* But data type will be same for both table.



|  |
| --- |
| **Create: student\_id int foreign key references emp(student\_id)**  **ALTER:** |

**Referenced table (base/parent table)**

* Insert (no violation)
* Delete (may cause as if we delete data from emp table and it is already part of another table that time it will create problem or it is not part of any table that time it will not make any problem)

If it may cause that time, we follow below rules.

* on delete cascade (It means selected row will be deleted from parent table and child or relevant table)
* on delete set null (The selected row will be deleted from parent table and on child table that value is is replaced by null value. If that value is primary key of child table that time it will not execute.
* On delete No action (No action will perform)
* Update (that time it will create problem or it is not part of any table that time it will not make any problem).

If it may cause that time, we follow below rules.

* ON update cascade (It means selected row will be update from parent table and child or relevant table)
* On update set null (The selected row will be update from parent table and on child table that value is replaced by null value. If that value is primary key of child table that time it will not execute.
* On update No action (No action will perform)

**Referencing table (child table)**

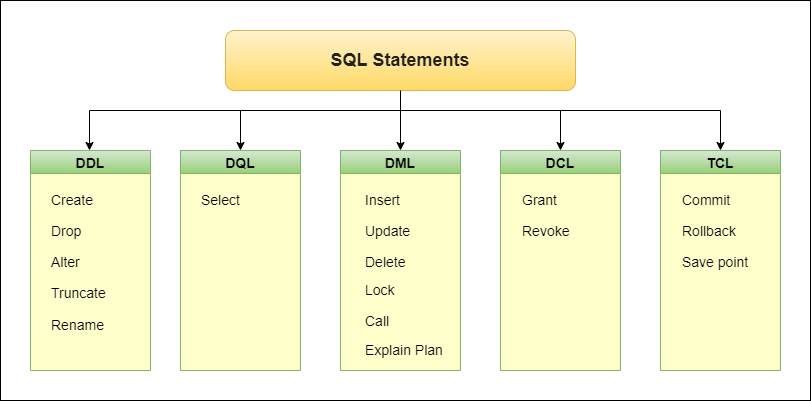
* Insert (It may cause violation)
* Delete (will not cause any violation)
* Update (May cause violation)

What are the 5 types of SQL commands?

**SQL Commands**

SQL commands are categorized into 5 categories.

* **DDL -**Data Definition Language
* **DQL -**Data Query Language
* **DML -**Data Manipulation Language
* **DCL -** Data Control Language
* **TCL -**Transaction Control Language



**DDL -**Data Definition Language

CREATE--->if we have to create anything in our database then we have to use Create command.

ALTER --->if we want to modify existing table then we can use ALter Command

alter table student add student\_marks int

DROP--->

Where we use drop:

* We can use drop to remove column from table.

Alter table (name) drop column (name)

* We can use drop to remove table from database

Drop table (name)

DQL - Data Query Language

SELECT---->if want to display the data what we have in our table then we can use select command. Example:

SELECT \* FROM VoterList

SELECT FirstName, LastName FROM VoterList

DML - Data Manipulation Language

Insert - to insert new records in a table.

String/Date-->Values can we pass with in a inverted comma.

DATE--->'10-03-2023'('yyyy-mm-dd')

Insert into employeeDetails(EmployeeID,FirstName,LastName,DepartmentId)

Values(23,'Suruti','roy', 45,34,'1995-10-18','2017-05-29',13000,'F')

Update - to modify the existing records in a table.

update employeeDetails set lastname='ghosh' where Gender ='F'

Delete -  is used to delete existing records in a table.

delete from employeeDetails where employeeid =12

**Table Manipulation**

Create (Table): Use to create new table

* create table er (emp int primary key,Emp\_naeme varchar (10))

Alter (Table): Add, delete, data type change in a column.

* + Add (COLUMN)--alter table EmployeeDetails add emp\_title varchar (20)
  + Update (COLUMN)---alter table EmployeeDetails alter column emp\_title varchar (2000)
  + Drop (COLUMN)--alter table EmployeeDetails drop column emp\_title
  + INSERT (CLOUMN)—insert into tables EmployeeDetails(name, age) values(‘uu’,56)

Insert into EmployeeDetails select

* + Update(record)— To update new record.
    - UPDATE EmployeeDetails SET LASTNAME='TA'
    - UPDATE EmployeeDetails SET LASTNAME='YU' WHERE FIRSTNAME='SURUTI'
    - UPDATE EmployeeDetails SET LASTNAME='XRU' WHERE FIRSTNAME='SURUI' AND GENDER='F'
  + DELETE (Record)- Delete the record.

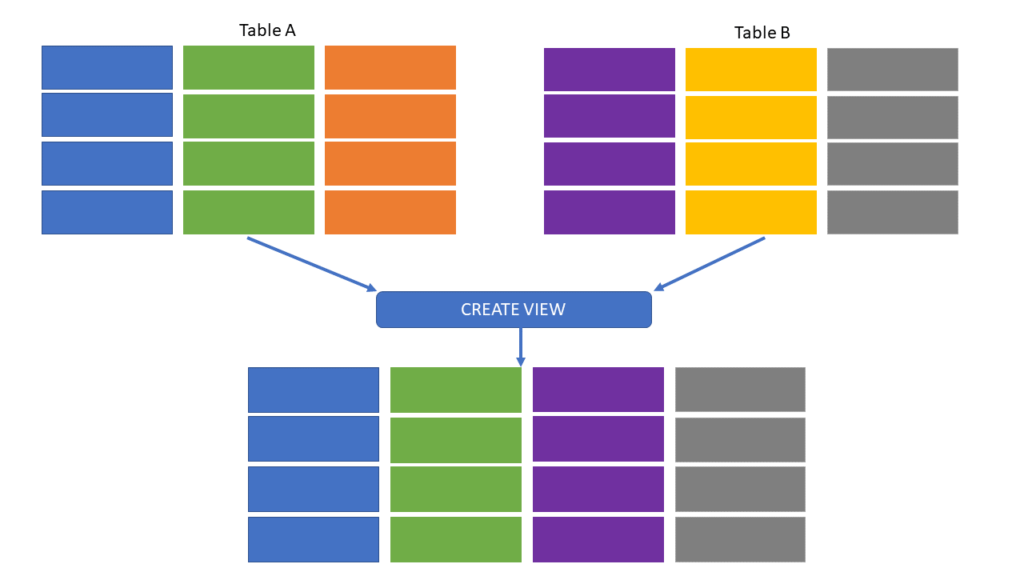
DELETE FROM EmployeeDetails WHERE Lastname='yu'

Drop (Table): It is used for drop table

* + - Drop table EmployeeDetails
    - drop table if exists varchar\_max\_test

VIEW TABLE:

Its contents are based on the base table. It contains rows and columns similar to the real table. In MySQL, the View is a **virtual table** created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own. The View and table have one main difference that the views are definitions built on top of other tables (or views). If any changes occur in the underlying table, the same changes reflected in the View also.



CREATE VIEW table with where clause:

create view vw\_wherE

as

select EmployeeID,FirstName,

from EmployeeDetails

where EmployeeID=16

EXECUTE TABLE: select \* from vw\_where

create view with join clause table:

CREATE view vw\_employeeDetails

as

select

EmployeeID,

FirstName,

lastName,

d.DepartmentName

c.ClientName

from EmployeeDetails as e

inner join department.d

on e.departnamentid=d.departnamentid

inner join client.c

on e.departnamentid=c.departnamentid

DROP VIEW TABLE:

drop view vw\_where

Replace with view :

**OPERATOR**

* SQL Arithmetic OPERATOR (+, -, \*, /, %)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Add | 20+30 |
| - | Subtract |  |
| \* | Multiply |  |
| / | Divide |  |
| % | Modulo |  |

SQL Bitwise OPERATOR (&, |, ^)

* SQL Comparison OPERATOR (=, <, >, <=, >=, <>)

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| <> | Not equal to |

* SQL Compound OPERATOR

|  |  |
| --- | --- |
| **Operator** | **Description** |
| += | Add equals |
| -= | Subtract equals |
| \*= | Multiply equals |
| /= | Divide equals |
| %= | Modulo equals |
| &= | Bitwise AND equals |
| ^-= | Bitwise exclusive equals |
| |\*= | Bitwise OR equals |

* + - * SQL Logical OPERATOR
* Suppose we have exact value to compare then we can use below command.
* SELECT \* FROM EmployeeDetails WHERE FirstName='Rohit'
* If we don’t have exact value but we have some patterns that time we will use like operator.
* The LIKE operator is used in the WHERE condition to filter data based on some specific pattern. It can be used with numbers, string, or date values. However, it is recommended to use the string values.
* SELECT \* FROM EmployeeDetails WHERE FirstName LIKE 'R%'
* % sign represent whatever value we have after the R that value will not compare with first name.
* a%--validate firstcharacter
* %a--validate lastcharacter
* %a%-check the a character is present or not in any position
* \_r%'-- check the "r" in the second position
* A\_%--- Finds any values that start with "a" and are at least 2 characters in length
* a\_\_% ---Finds any values that start with "a" and are at least 3 characters in length
* a%o-- Finds any values that start with "a" and ends with "o"
* h\_t finds hot, hat, and hit
* h[oa]t finds hot and hat, but not hit
* h[^oa]t finds hit, but not hot and hat
* c[a-b]t finds cat and cbt

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Clause** | **Example** |
| ALL | The ALL operators allow to perform a comparison between a single column value and a range of other values | SELECT, WHERE and HAVING | SELECT ProductName  FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10) |
| ANY | The ANY operators allow to perform a comparison between a single column value and a range of other values | SELECT, WHERE and HAVING,comparison operator | SELECT \* FROM Products WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50) |
| SOME | SOME must match at least one row in the subquery | SELECT, WHERE and HAVING | SELECT \* FROM Products WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20) |
| EXISTS | The EXISTS operator is used to test for the existence of any record in a subquery | where clause | SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20) |
| BETWEEN | The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates | where clause | SELECT \* FROM Products WHERE Price BETWEEN 50 AND 60; |
| IN | It is pulling different row value in same column. | where clause | Select \* from products where city in ('india', 'uk') |
| LIKE | The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. | where clause | SELECT \* FROM Customers WHERE City LIKE 's%' |
| AND | When we pull different record in different column that time it is used. | where clause | SELECT \* FROM Customers WHERE City = "London" AND Country = "UK"; |
| OR | When we pull different record in different column or same column. | where clause | SELECT \* FROM Customers WHERE City = "London" OR Country = "UK"; |
| NOT | Displays a record if the condition(s) is NOT TRUE | where clause | SELECT \* FROM Customers WHERE City NOT LIKE 's%'; |
| Is null | It is not possible to test for NULL values with comparison operators. That time we use this operator | where clause | SELECT column\_names FROM table\_name WHERE column\_name IS NULL |
| Is not null | It is not possible to test for NULL values with comparison operators. That time we use this operator | where clause | SELECT column\_names FROM table\_name WHERE column\_name IS NOT NULL |
| Wildcard Character | It is used to substitute one or more characters in a string | LIKE operator | SELECT \* FROM Customers WHERE City LIKE '\_ondon' |

* + - * SQL SET Operator

A SET operator in SQL combines the results of two or more SELECT statements or compares rows from two or more tables. It returns a single result set by eliminating duplicate rows.

* SELECT column1, column2
* FROM table1
* UNION/UNION ALL/INTERSECT/EXCEPT
* SELECT column1, column2
* FROM table2;
* The number of columns in the SELECT statement on which you want to apply the SQL set operators must be the same.
* The order of columns must be in the same order.
* The selected columns must have the same data type.
* If you want to order/sort the results, the ORDER BY clause must go at the end of the last query.
* You can't add ORDER BY inside each SELECT query before the set operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| UNION | Returns all distinct rows selected by either query, eliminating duplicates | select EmployeeID,DepartmentID from EmployeeHistory union select EmployeeID,DepartmentID from Employee\_nes |
| UNION ALL | Returns all rows selected by either query, including duplicates | select EmployeeID,DepartmentID from EmployeeHistory union all select EmployeeID,DepartmentID from Employee\_nes |
| INTERSECT | Returns returns only the rows common to both queries | select EmployeeID, DepartmentID from EmployeeHistory intersect  select EmployeeID,DepartmentID from Employee\_nes |
| EXCEPT | Returns only the rows unique to the first query | select EmployeeID, Department from EmployeeHistory except select EmployeeID,DepartmentID from Employee\_nes |

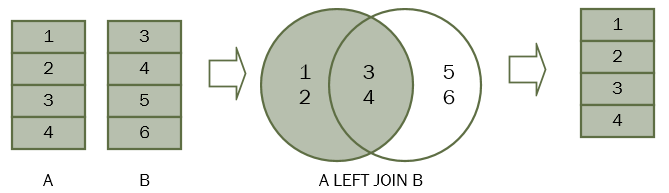
JOIN

 A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them

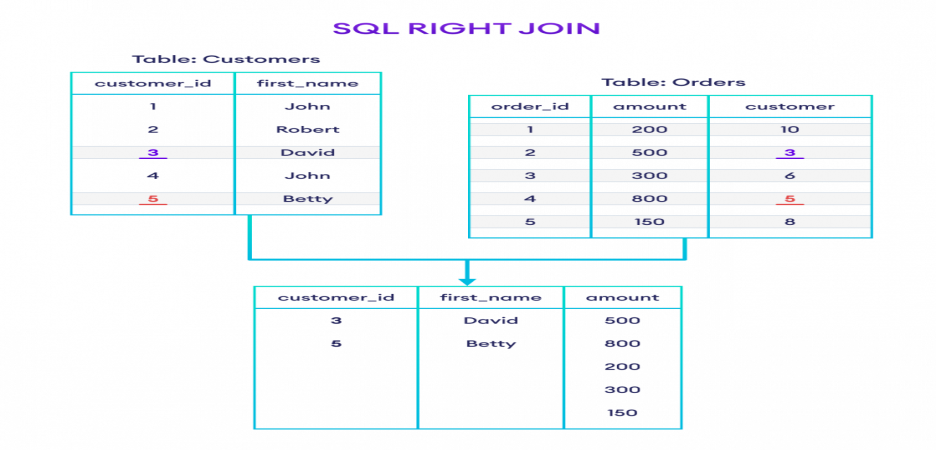
* INNER JOIN--->*Will return those data which is present in both the table*



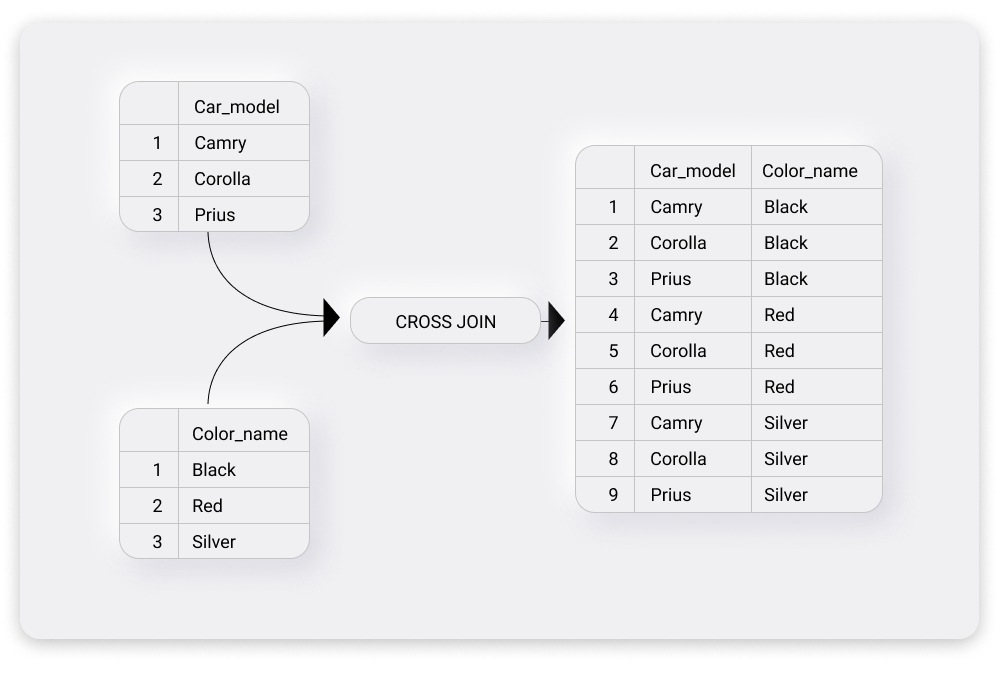
LEFT JOIN --->*LEFT JOIN WILL RETURN all the data from LEFT TABLE and matching record from the right table*

**

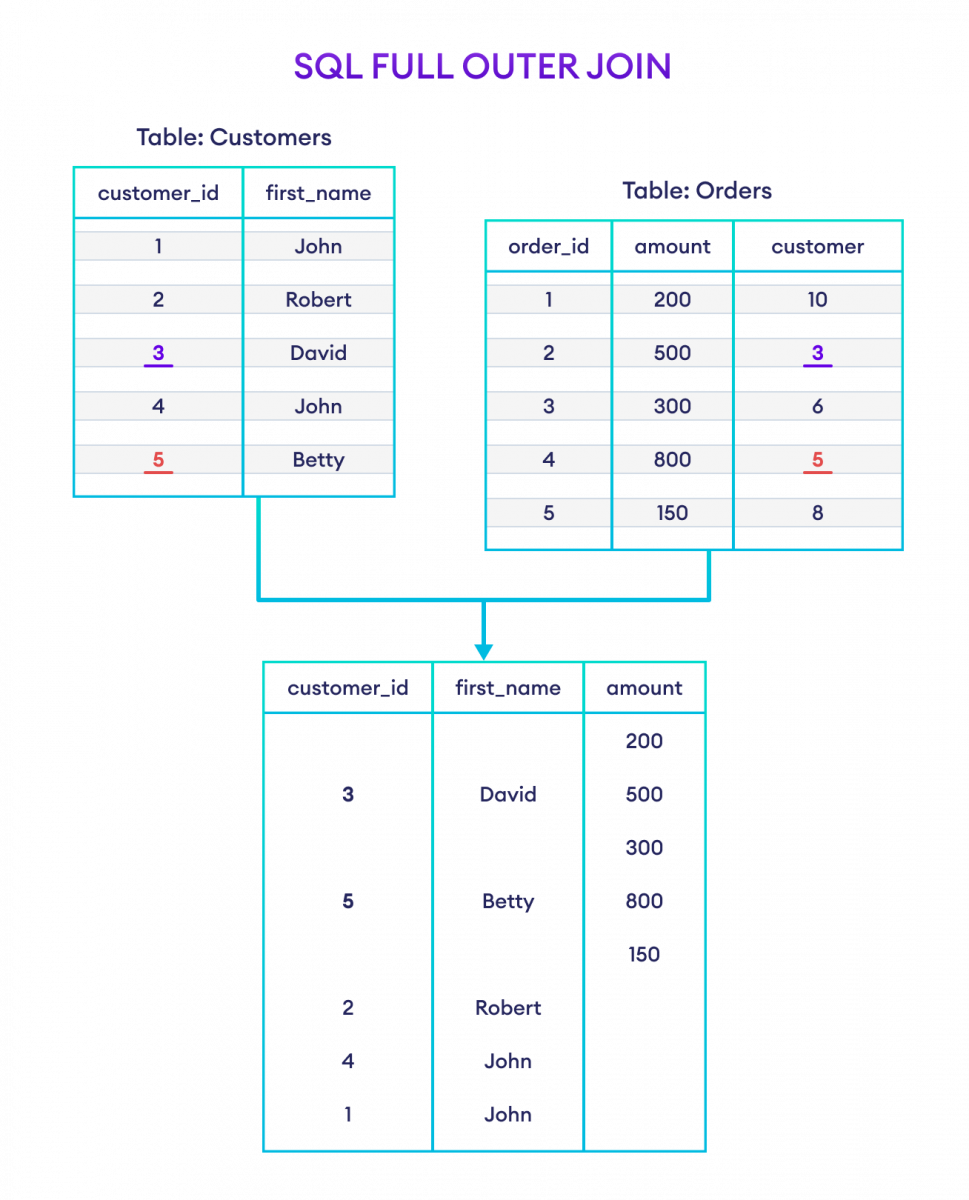
RIGHT JOIN --->*it will return all the data from right table and matching record from the left table*



CROSS JOIN ---> *CROSS JOINs are used to combine each row of one table with each row of another table, and return the Cartesian product of the sets of rows from the tables that are joined.*



* FULL OUTER JOIN



**================================================**

Function

SQL Server has many built-in functions.

* String Functions
* Date Time Functions
* Numeric Functions
* Conversion Functions
* Advanced Functions

SQL Server String Functions

* **UPPER()à** This function will take a parameter and will return the value into the capital letter.
* **LOWER ()à** This function will take a parameter and will return the value into the small letter.
* **LEFT ()à** This function will take two parameters.
  + - * STRING
      * HOW MANY Characters you want to fetch.
* **RIGHT ()à**This function will take two parameters.
  + - * STRING
      * HOW MANY Characters you want to fetch.
* **REPLACE ()à** Replaces all occurrences of a specified string with another string value. This function will take two parameters.
  + - * STRING
      * Which character or value wants to replace.
      * Which character or value wants?
* **LTRIM()à** Remove left blank space from the string
* **RTRIM()à** Remove right blank space from the string
* **TRIM ()à** Remove all right and left side the blank space from the string.
* **SUBSTRING ()à** Returns part of a character, binary, text, or image expression. This function will take two parameters.
  + - * STRING
      * starting Index(from which index I want to see)
      * How many character we want

* **REVERSE ()à** Return charter on reverse way from string.
* **CHARINDEX ()à** Searching character in a string and that character is found then give a index value if not found then return null. This function will take two parameters.
  + - Which character is tried to find out.
    - In which string it is available.

SQL Server Date Time Functions

* **YEAR ()à**SELECT DOB,YEAR(DOB) FROM EmployeeDetails
* **MONTH ()à**SELECT DOB,MONTH(DOB) FROM EmployeeDetails
* **DAY ()à**SELECT DOB,DAY(DOB) FROM EmployeeDetails
* **DATENAME ()à**DAYNAME()/Mnthname
  + - * SELECT DOB,DATENAME(MONTH,DOB)FROM Employee\_import
      * SELECT DOB,DATENAME(WEEKDAY,DOB)FROM Employee\_import
* **DATEADD ()**àThis function will take two parameters.
  + Interval(dd/mm/yy)
  + How many no of days/month/year we want to add.
  + The data value
    - * Select dob,dateadd(yy,3,dob)FROM Employee\_import
      * Select dob,dateadd(mm,3,dob)FROM Employee\_import
      * Select dob,dateadd(dd,3,dob)FROM Employee\_import

**DATEDIFF ()**àThis function will take three parameter

Interval

Start date

end date

**GETDATE ()à** It will return current date

SQL Server aggregate functions:

Aggregate function is used to group the data. An aggregate function in [SQL](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql) performs a calculation on multiple values and returns a single value.

* **Sum()**
  + - * SELECT SUM(SALARY) FROM Employee\_import
      * SELECT DEPARTMENTID,CLIENTID,SUM (SALARY)FROM Employee\_import GROUP BY DEPARTMENTID,CLIENTID
* **Avg()**
  + - SELECT MIN(SALARY) FROM Employee\_import
    - SELECT DEPARTMENTID,CLIENTID,AVG(SALARY)FROM Employee\_import GROUP BY DEPARTMENTID,CLIENTID
* **Min()**

SELECT MIN(SALARY) FROM Employee\_import

SELECT DEPARTMENTID,CLIENTID,MIN(SALARY)FROM Employee\_import GROUP BY DEPARTMENTID,CLIENTID

* **Max()**
  + - * SELECT MAX(SALARY) FROM Employee\_import
      * SELECT DEPARTMENTID,CLIENTID,MAX(SALARY)FROM Employee\_import GROUP BY DEPARTMENTID,CLIENTID
* **Count()** 
  + - * SELECT COUNT(\*) FROM Employee\_impot
      * SELECT DEPARTMENTID,CLIENTID,MAX(SALARY)FROM Employee\_import GROUP BY DEPARTMENTID,CLIENTID

SQL Server Window functions:

**a)-ROW\_NUMBER()** It will filter the row basis on the mentioned column. And it provides unique row no.

select\*,ROW\_NUMBER()over(order by salary asc) as rowed From Employee\_import

**b)-RANK()** It is also provide row no. But suppose any duplicate value is there then it will show duplicate row no. when different value will come on selected column then it will jump different row id.

Select rank() over(order by salary asc) as rankid,\* from Employee\_import

**c)-DENSE\_RANK()**

**d)-NTILE()**

USER Defined Function

There are two main types of user-defined functions in SQL based on the data they return:

1. **Scalar functions:** These types of functions return a single value, i.e float, int, varchar, datetime, etc.
2. CREATE FUNCTION scalar\_func
3. (
4. @a AS INT, -- parameter a
5. @b AS INT -- parameter b
6. )
7. RETURNS INT -- return type
8. AS
9. BEGIN
10. RETURN @a + @b -- return statement
11. END;
12. **Table-Valued functions:** These functions return tables.

Having Clause:

**HAVING** is used to filter groups. **WHERE** is used to filter rows. It filters out the unwanted data records. SELECT is used to filter columns. Having clause always contain

Aggregate function.

|  |  |
| --- | --- |
| **HAVING** | **WHERE** |
| 1.The HAVING clause is used to fetch the data from the groups according to the given condition. | 1. The WHERE clause is used in the data from the tables according to the given condition. |
| 2. The HAVING clause is always executed with the GROUP BY clause. | 2. The WHERE clause can be executed without the GROUP BY clause. |
| 3. The HAVING clause can include SQL aggregate functions in a query or statement. | 3. We cannot use the SQL aggregate function with WHERE clause in statements. |
| 4. We can only use SELECT statement with HAVING clause for filtering the records. | 4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements. |
| 5. The HAVING clause is used in SQL queries after the GROUP BY clause. | 5. The WHERE clause is always used before the GROUP BY clause in SQL queries |
| 6. We can implement this SQL clause in column operations. | 6. We can implement this SQL clause in row operations. |
| 7. It is a post-filter. | 7. It is a pre-filter. |
| 8. It is used to filter groups. | 8. It is used to filter the single record of the table. |

#######Store Procedure

A stored procedure is a prepared SQL code that you can saved in database, so this code we can be used multiple times. So, if we have an SQL query that we write multiple times, save it as a stored procedure, and then just call it to execute it.

Based on the statements in the procedure and the parameters you pass, it can perform one or multiple DML operations on the database, and return value, if any.

1. Reusable: As mentioned, multiple users and applications can easily use and reuse stored procedures by merely calling it.
2. Easy to modify: we can quickly change the statements in a stored procedure and when we want to, with the help of the ALTER TABLE command.
3. Security: Stored procedures allow you to enhance the security of an application or a database by restricting the users from direct access to the table.
4. Low network traffic:  A stored procedure reduces network traffic between the application and the database server, resulting in increased performance. It is because instead of sending several [SQL](https://www.javatpoint.com/sql-tutorial) statements, the application only needs to send the name of the stored procedure and its parameters.

Types of Stored Procedures

* User-defined Stored Procedures: This procedure is developed by developer.

**CREATE** **PROCEDURE** [schema\_name].procedure\_name

                @parameter\_name data\_type,

               @parameter\_name data\_type

**AS**

**BEGIN**

      -- SQL statements

      -- SELECT, INSERT, UPDATE, or DELETE statement

**END**

* System Stored Procedures: This procedure is developed by the servers administrative.

Index:

The Index in SQL is a special table used to speed up the fetching data in the database tables.

Indexes are used to retrieve data from the database more quickly than otherwise.

* TABLE SCAN--> it will check each and every page from the tables
* INDEX SCAN-->it will check the index page
* INDEX SEEK-->when sql server have a exact idea/exact location where the data is stored on time sql will perform index seek

---Merge table

The merge statement selects the rows from one or more than one tables (called source table) and based on conditions specified, insert or update data to another table (called target table).

As MERGE statement in SQL is the combination of three insert, delete, and update statements. So, if there is a **Source table** and a **Target table** that are to be merged, then with the help of MERGE statement, all the three operations (INSERT, UPDATE, DELETE) can be performed at once.

-- For Inserts

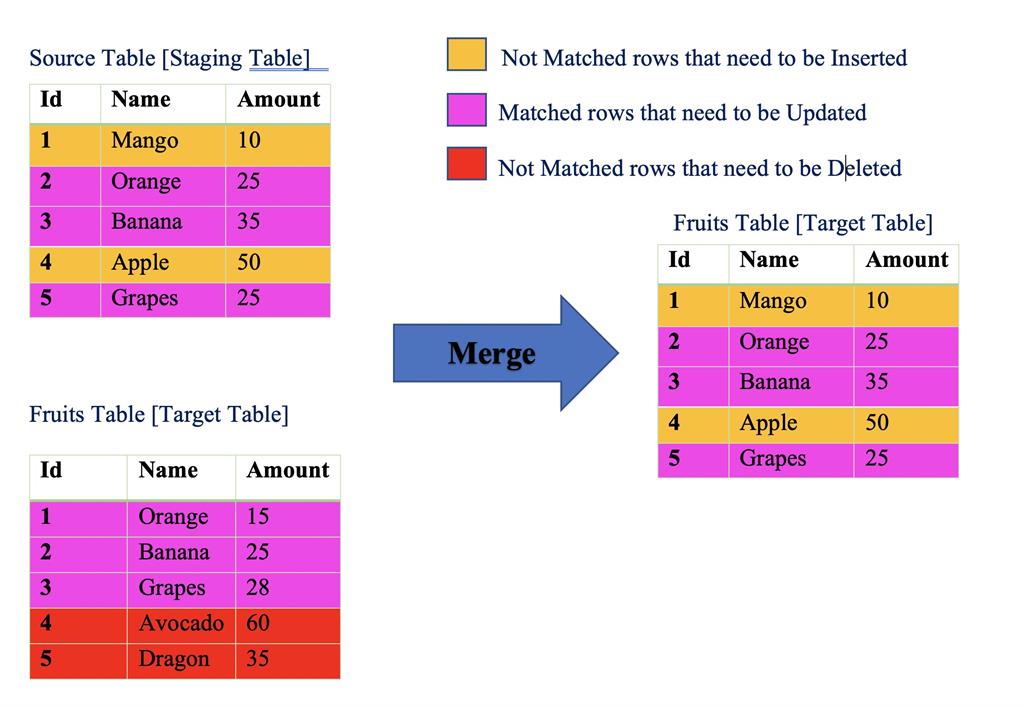
>those products which are present in the source table but not available in the target then we have to insert that product in target table.

- For Updates---

-Here when the productid got matched from source and target then it will update the ProductName and price column in our target table.

---FOR DELETE

it will check for those products which are present in target table but not present in source table then it will delete those products from our target table.



HAVING is used to filter groups. WHERE is used to filter rows. It filters out the unwanted data records. SELECT is used to filter columns.

Explanation: We can update only a single table at a time using UPDATE. INSERT INTO is used to insert records, SELECT is used to retrieve data; DELETE is used to delete records.

Problem:

1.Can not add primary key, default constraint on existing table.

2. and, any all, exist,some, operator not use in join

3.What is scehma

4. This query is not working. --I want to fetch data maximum salary departmnet wise there salary will be 50000 and dispaly there names

select firstname from employee\_details

where firstname in

(select DepartmentID from employee\_details group by DepartmentID having max(salary)>70000)