**SQL:**

🡪Structured query language which is a computer language for storing, manipulating, and retrieving the data stored in relational database management systems(rdbms).

🡪It is an open source rdbms that stores data in a structed format using rows and columns. It creates database to store and manipulate the data.

🡪To perform various operations user, make request by typing specific statements.

🡪The server responds to information from the user and display it on the user side.

* **Constraints:**

-🡪 constraints in SQL means we are applying certain conditions or restrictions on the database. This further means that before inserting data into the database, we are checking for some conditions.

-🡪If the condition we have applied to the db. holds true for the data which is to be inserted, then only the data will be inserted into the database tables.

**Types of Constraints**

* Not Null
* Unique
* Check
* Primary Key
* Foreign Key
* **Null:**

🡪Null is nothing, it represents unknown value.

-🡪Null does not occupy space in the memory.

🡪Null is neither zero nor a blank space.

🡪If it performs any arithmetic operations with null it reserves null itself. Two nulls never be same in oracle.

* **Not Null:(opposite to null)**

🡪Not Null ensures some value should be present in a particular column in a table.

**Example: Create a table student with the fields id and name as not null.**

Create table student (

Std id int not null,

Name varchar (20) not null,

Address varchar (20)

);

* **Unique**

**🡪**This **constraint** helps to uniquely identify each row in the table i.e., for a particular column, all the rows should have unique values. We can have more than one unique column in a table.

* It does not accept one duplicate value.
* It can accept multiple null values.

**Example:**

**Creates a table student where the field Id is specified as unique. i.e., no two students can have the same Id.**

Create table student (

Std id int not null unique,

Name varchar (20),

Address varchar (20)

);

**Create a student table and apply a UNIQUE constraint on one of the table's columns while creating a table.**

1. **CREATE** **TABLE** student (StudentID **INT** **UNIQUE**, Student\_FirstName **VARCHAR**(20), Student\_LastName **VARCHAR**(20), Student\_PhoneNumber **VARCHAR**(20), Student\_Email\_ID **VARCHAR**(40));
2. **Create a student table and apply a UNIQUE constraint on more than one table's column while creating a table.**
3. **CREATE** **TABLE** student (StudentID **INT**, Student\_FirstName **VARCHAR**(20), Student\_LastName **VARCHAR**(20), Student\_PhoneNumber **VARCHAR**(20), Student\_Email\_ID **VARCHAR**(40), **UNIQUE**(StudentID, Student\_PhoneNumber));

**Syntax to apply the Unique Constraint on an existing tables’ column:**

**Alter table Table Name add Unique (Column name);**

Consider we have an existing table student, without any constraints applies to it. Later, we decide to apply a unique constraint to one of the tables columns. Then we will execute the query.

**Alter table student ADD Unique(studentId);**

* **Primary key:**

🡪Primary key is a column which uniquely identifies a record.

🡪It is a combination of not null & unique constraint.

🡪Creation of primary key is highly recommended but not mandatory.

🡪In a table we can have only one primary Key.

**🡪syntax: create table Table Name(columnname1 datatype Primarykey,columnname2 datatype,--)**

Create a student table and apply the primary key constraint while creating a table.

Create table student(

studentId int primary key,

name varchar(20),

Address varchar(20)

);

* **Foreign Key:**

🡪Foreign key is a column which creates the relationship between two tables.

🡪It is also called referential integrity constraint. It can accept null and duplicate values.

🡪 can have more than one foreign key in a table.

🡪Foreign key is created in a child table.

🡪If we have foreign key in a table, then the same column should be a primary key in master table.

Syntax: create table table name(

columnname1 datatype(size)primary key,

columnnameN datatype(size),

foreign key(column Name) references

parent-table-name(primary-key-columnname)

);

**Create an employee table and apply the foreign key constraint while creating a table.**

To create a foreign key on any table first we need to create a primary key on a table.

Create table employee(empId int not null primary key,

name varchar(20),

salary int

);

**We will create a query to apply a foreign key on the department table referring to the primary key of the employee table,i.e.emp\_ID.**

Create table department(deptid int not null primary key,

Deptname varchar(20),

Empid int not null ,

foreign key(empid)references employee(empid));

* **Check Constraint:**

🡪whenever a check constraint is applied to the tables column and the user wants to insert the values in it, then the value will first be checked for certain conditions before inserting the value into that column.

🡪Check constraint is used to provide a additional validation according to the customer requirement specification.

**Syntax:create table tablename(columnname1 datatype check(columnname1 condition),columnname2 datatype…. columnnameN datatype);**

**Create a student table and apply check constraint to check for the age less than or equal to 15 while creating a table.**

Create table student(

Studentid int,

Student FirstName varchar(20),

Student lastName varchar(20),

Age int check(Age <=15)

);

**Apply multiple constraints on multiple columns.**

**Create a student table and apply check constraint to check for the age less than or equal to 15 and a percentage greater than 85 while creating a table.**

Create table student (

Studentid int,

Student FirstName varchar (20),

Student lastName varchar (20),

Age int ,

percentage int,

check(Age <=15 and percentage >85)

);

**Syntax to apply check constraint on am existing column.**

* **Alter table table name add check (column name condition);**
* Ex: Alter table student add check (Age<=15);

**Functions or subset commands**

* **Dql**

Select: it is used to retrieve the data from the database.

* DDL
* Create: it is used to create a table in db.
* Alter: it is used to modify the structure of the existing table. it means we can change the table name, we can add a row , add a column,
* delete a column or rename the column name.
* rename: It is used to rename the table name.
* Truncate: it is used to delete all the records permanently, but the structure of the table remains same.
* Drop: Drop command is used to delete the entire table.
* Create: how to create a table for employee

create table Employee (

EmpId int,

firstname varchar (20),

lastname varchar (20),

Email varchar (20),

Phone no varchar (20),

salary int

);

Create an employee table with primary key as empid

create table Employee (

EmpId int primary key,

firstname varchar (20),

lastname varchar (20),

Email varchar (20),

Phone no varchar (20),

salary int

);

* Alter:it is used to modify the structure of the existing table. it means we can change the table name, we can add a row , add a column, delete a column or rename the column name.

Syntax:

ALTER TABLE table\_name

ADD column\_name1 data\_type,

column\_name2 data\_type,

...

* add a column name into employee table.

Alter table Employee

Add Address Varchar (30),

City varchar (30),

Pin code int;

* Rename the column name

Syntax: ALTER TABLE table\_name

RENAME COLUMN old\_column\_name TO new\_column\_name;

* alter table employee rename column phone no To Mobile.
* Rename the column size from employee table
* alter table employee alter column EmailId varchar (20);
* add the column name for employee.
* alter table employee add City varchar (30);
* Drop column: It is used to delete the columns one or more columns.
* Syntax: Alter table employee drop column city;
* Rename: It is used to rename the table name
* Syntax: Alter table employee rename To Emp;
* Drop:it is use to delete the entire table
* Syntax: drop table Tablename
* Drop table student;

Truncate: it is used to delete all records permanently, but the structure of the table remains same.

* Syntax: Truncate table student or

Delete from student.

* DML:
* Insert: insert command is used to insert the records into the table
* Syntax: insert into table name (column name1, column name2,…) values(value1, values2,---);
* Ex:insert into employee(empid,firstname,lastname,emailed, phoneno,salary,address, city, pincode)values(20,’lucky’,’man’,’luckyman@gmail.com’,999999999,30000,’hyderabad’,’Gachibowli’,500032);

Update:update command is used to modify the data in a table.

* Syntax:Update tablename

Set column name1=new value1

Set column name2=new value2

Where [clause]

* Ex: Update employee

Set salary=35000

Where EmpId=21;

* Update employee

Set FirstName-‘blakely’, lastname=’charly’

Where EmpId=24;

* Update employee

Set EmailId=Replace(EmailId,’@gmail.com’,’@outlook.com’)

Where EmpId=23;

* Update employee

Set PhoneNo=9898989898

Where EmpId=20;

* Update employee

Set salary=salary+(salary\*10/100);

Delete:delete command is used to delete the selective area

Syntax:Delete from tablename [where condition]

* Delete from employee where EmpId=4

It will delete the 4th record of the table.

* Delete from employee where salary>50000.
* Delete from Employee; --🡪 it will delete all the records of the table.

Operators:

* **Logical Operators:**
* AND & OR: In sql and & or operators are used for filtering the data and getting precise result based on the conditions. The SQL and & or operators are also used to combine multiple conditions. These two operators can be combined to test for multiple conditions in a select, insert, update, delete statement.

🡪These two operators are used with the where clause.

🡪And operator is used when all the conditions are true

🡪Syntax: select \* from table name where condition1 and condition2 and …conditionN;

🡪OR operator is used if any one of the conditions are true.

🡪Syntax: select \* from table name where condition1 or condition2 or ….conditionN;

🡪Fetch the records from the emp table where salary is 38500 and address is cyderabada by using AND condition.

* Select \* from employee where salary=385000 and address=’Cyberabad’

By using OR condition:

* Select \* from employee where FirstName=’Rocky’ or FirstName=’Lucky’
* select\* from employee where FirstName=’Lucky’ or Last Name=’Man’;

NOT:<>

* Select column1 column2 from table name where not condition.

🡪The following SQL statement selects all fields from “Customers” where country is not “UK” SELECT \* FROM Customers WHERE NOT Country=’UK’;

* Select\* from customer where not country=’UK’;
* Select\* from customer where not country=’UK’ and not country=’USA’;
* Select\* from customer where country <> ’India’;
* **Operators:**

🡪We can use various arithmetic operators on the data stored in the tables.

Arithmetic operators are:

* **+** : The addition is used to perform an addition operation on the data values.
* **-:** This operator is used for the substraction of the data values.
* **/** :This operator works with the ‘ALL’ KEYWORD AND IT CALCULATES DIVISION OPERATIONS.
* :This opertaor is used for multiplying data values.
* **%:**Modulus is used to get the remainder when data is divide by another.
* **Comparison operators:**

Comparison operators which is used to compare one expression value to other expression .The following are the comparison operators.

= ,<,>,>=,<=,<>(not equal to).

Select \* from maths where marks=50;

* **Special Operators:**
* **IN, IS, LIKE, BETWEEN.**
* **IN:**In operators is used instead of multiple values. (Or) In operator works same as that of operator.

🡪IN operator checks a value within a set of values separated by commas and retrieves the rows from the table that match.

* **Syntax: Select <column-list>**

From <Table-name>

Where column-Name IN (v1,v2,v3---vn);

🡪Ex: w.a.q. to list all employees for working in 10,20,30 in deptno table

🡪Select \* from emp where deptno IN(10,20,30);

🡪Select empid, firstname, lastname, salary from emp where emid in (1,3,5,6);

* **Sub-query with in Operator**

We can use the subquery within operator that returns record from the single column.the subquery cannot include more than one column in the select column list.

Select empid,firstname, lastname,deptid

From employee

Where deptid in(select deptid from department WHERE DEPTID >2);

* **Not In**

Use the not operator with the in operator to filter record that do not fall in the specified values.

Select empid,firstname,lastname,salary from emp where empid not in(1,3,5);

* **Is: To compare with null we use Is Operator.**

Select <column-List>

From <table-name>

Where column-name is null;

* **Like**: Like operator is used to perform pattern matching to find the correct result. It is used in insert, select, update, delete statement with the combination of where clause.

Syntax: select <column-list>

From <table name>

Where column-name like pattern{[escape] escape-character}

**|**

Optional

* **Escape**-**character**: it is a optional. It allows you to test for literal instances of a wildcard character such as **“%** “ or **“\_** “.If you don’t provide the escape\_character, MySQL assumes that “\” is the escape\_character.

Inside a pattern we are using two wild cards. They are

1. **%** (**Percentage**): It matches with a n.no of characters.

**EX:** select off-name from officers where address like ‘Lucknow%’;

**EX**: select off-name from officers where address Not like ‘Lucknow%’;

**EX**: select off-name from officers where address like ‘luck%’;

**2.\_(Underscore**):It matches with single character.

**Ex**: select off\_name from officers where like ‘luc\_now’;

**Stored Procedure:**

A stored procedure is group of T-SQl(transact sql) statements. If you have a situation, where you write the same query over and over again, we can save that specific query as a stored procedure and call it just by its name.

To execute stored procedures:

1.spGetEmployees

2.ExECspGetEmployees

3.Execute spGetEmployees (or)

We can also right click on the procedure name, in object explore in SQL server management studio and select execute stored procedure.

OR

A stored procedure is nothing more than a piece of code that performs some repetitive set of actions.

🡪It performs a particular task by executing a set of actions or queries against the database.

🡪The code for stored procedure is stored in the database and can be executed at any time.

🡪Stored procedures are typically used to insert your records into one of more tables, update or delete data from tables, and to generate reports via the select statement.

🡪Its possible for a stored procedure to do more than one thing. If we might want to run an update statement which modifies some information, then run a select statement to return the updated statistical information.

🡪Stored procedures are useful because they allow the developer to write code once, then execute it many times.

🡪Its possible to build up a library of stored procedures for a database overtime, all of which provide a programming interface to the database tables.

Ex: we have existing employee table for that again we will execute the query with procedure condition.

1.Select \* from employee which consist of empid,name, salary,email.

Instead of that what we can actually do is wrap this query inside a stored procedure and call that s.p rather than we having to write this query again and again.

1.1.Select name,empid,salary from employee

To create a stored procedure we use procedure command.

* **Create procedure:**

**Syntax**: **create procedure statement.**

**It is used to create a new stored procedure.**

Create procedure <name of procedure>

(parameters (optional))

As

Begin;

// Statements here….

End;

* **Ex**:Create procedure spemploye

As

Begin

Select empid,name,salary from employee

End.

In Object explorer-🡪 master database🡪programability🡪stored procedures🡪spemploye.

To execute the table:spemployee

* **Alter Procedure:**

**Syntax: Alter Procedure statement**

**It is used to alter existing stored procedure.**

Alter procedure <name of procedure>

(parameters (optional))

As

Begin;

// Statements here….

End;

* **Drop Procedure:**

**Syntax**: **Drop** **Procedure statement**

**It is used to remove existing stored procedure.**

**Drop procedure [if exists] <name of procedure>;**

**Stored procedure with parameter**

**Create procedure spgetemployeesbynameanddept**

**@name varchar(20),**

**@deptid int**

**As**

**Begin**

**Select name,sal,deptid from employee where name=@name and deptid=@deptid**

**End.**

**In the above query @name and @deptid are parameters.**

**To execute the sp or above query: spgetemployeesbynameanddept ‘john’,23. Or spgetemployeebynameanddept @name=’john’ ,@deptid=23**

**Here john and 23 are we passed the parameters ,whatever we declared in query .**

* **View the procedure:**

To view the procedure -🡪programability🡪procedure table(spemployee)🡪 as sp🡪create right click🡪scripted to🡪new window.

( Or)

Sp\_helptext spemployee

🡪once we created a stored procedure ,if we want to change or modify the sp implementation.

Ex: we want to execute the name in alphabetic order or deptid in increasing order we can use alter statement by implementing order by.

Alter procedure spemployee

As

Begin

Select name,salary,email from employee order by salary

End.

* Delete the stored procedure

To delete the table we use drop command.

Drop procedure spemployee (Or) drop proc spemployee.

It will delete spemployee folder.

* Encrypt the text of the stored procedure by using the command with encryption and alter

**Alter procedure spgetemployeesbynameanddept**

**@name varchar(20),**

**@deptid int**

**With Encryption**

**As**

**Begin**

**Select name,sal,deptid from employee where name=@name and deptid=@deptid**

**End.**

* Store procedures -output parameters.

To create an sp with output parameter, we use the keywords out or output

Create procedure spgetemployeecountbygender

@Gender navrcahr(20),

@employeecount int output----------🡪 here we declared employee count as output parameter

As

Begin

select @employeecount=count(id)

from table employee

where gender=@gender

end

* **To execute the stored procedure without parameter**

**When sp something returns back to you, we want to hold it somewhere in a variable that’s why first we create a variable(as @totalcount) to receive the value.so now we are creating a variable @totalcount, and notice that the data type of variable must match with output parameter datatype.**

Declare @totalcount int

Execute spgetemployeecountbygender ‘male’,@totalcount output

Print @totalcount.

If you don’t specify the output keyword,when executing the stored procedure , the @employeetotal varaiable will be null.

Declare @totalcount int

Execute spgetemployeecountbygender ‘male’,@totalcount

If(@totalcount is null)

Print ‘@totalcount is null’

Else

Print ‘@totalcount is not null’

* **Group by clause:** It is used to group a selected set of rows into a set of summary rows by the values of one or more columns or expressions. It is always used in conjunction with one or more aggregate functions**.**

**(OR)**

* Group by clause is used in select statement to collect the data across multiple records and group the results by one or more columns.

The Group by statement is used for organizing similar data into groups.

🡪The **select** statement is used with the **group** **by** clause in SQL.

🡪**where** clause is placed before the **group** **by** clause in SQL.

🡪**order** **by** clause is placed after the **group** **by** clause in SQL.

**Syntax:select column1,function\_name(column2)**

**From table-name**

**Where condition**

**Group by column1,column2**

**Order by column1,column2;**

**Function\_name:table name.**

**Condition:which we used.**

* **Ex:waq to display all the records of the employee table but group the results based on the age column.**

🡪Select\* from employee group by Age;

🡪We have an existing table as a person details or employee table which consists of id,name,gender,salary,city.

🡪Select city,sum(salary) as totalsalary--🡪( here we want the data of city and salary so we need to implement group by command with some data otherwise it will get error)

From employee

Group by city.

**Filtering groups:Grouping by Multiple columns**

**🡪To filter the rows we can use where clause.**

**🡪Where clause is used to filter rows before aggregation,where as having clause is used to filter groups after aggregation.The following 2 queries produce the same result**

Select city, gender,sum(salary) as totalsalary

From employee

Group by city,gender.

Order by city(here we are declaring order by because it will give the result the city names in order by alphabetic order)

🡪If we want total no of employees from employee table

* **Multiple aggregate Functions(like count ,total,sum)**

🡪Select count(\*) from employee

🡪Select city, gender,sum(salary) as totalsalary,count(id) as (total employees)

From employee

Group by city,gender.

🡪If we want to see only particular rows result like for (gender only female people)

So here we are using where clause because to filter the rows

* **Where Clause:**

🡪Where keyword is used for fetching filtered data in a result set. It is used to fetch data according to particular criteria. **Where** keyword can also be used to filter data by matching patterns.

* **Syntax:**

**Select column1,column2 from table-name where column-name operator value;**

Ex: create a table employee.

Create table employee(

Empid int primarykey,

Name varchar(50),

Country varchar(50),

Age int(2),

Mob int(10)

);

Insert into employee(empid, name, country, age, mob)values (1,’rocky’,’uas’,25,99999999);

* **Where clause with logical operators:**
* **To fetch the records of employee with ages equal to 24.**

🡪Select \* from employee where age=24;

* **To fetch the empid, name and country of employees with age greater than 21.**

🡪Select empid, name, country from employee where age>21;

* **Where clause with between operator:**

🡪It is used to fetch filtered data in a given range inclusive of two values.

* **Syntax:**
* **Select column1,column2 from table-name**
* **Where column-name between value1 and value2;**

**Ex: To fetch the records of employees where age is between 22 and 24.**

🡪Select \* from employee where age between 22 and 24.

* **Where clause with like operator:**

🡪It is used to fetch filtered data by searching for a particular pattern in the where clause.

* **Syntax:**

**Select column1,column2 from**

**Table-name where column-name like pattern;**

* **To fetch the records of employees where name starts with the letter S.**

🡪Select \* from employee where name Like ‘s%’;

* **To fetch records of employees where name contains the pattern M**

🡪Select\* from employee where like ‘%m%’;

* **Where clause with In Operator:**

🡪It is used to fetch the filtered data same as fetched by=operator just the difference is that here we can specify multiple values for which we can get the result set.

* **Syntax:**

**Select column1,coulumn2 from table-name where column-name in (value1,value2,)**

* **To fetch the names of employees where age is 21 or 23.**

🡪Select name from emp1 where age in(21,23);

* **Waq to delete an employee record where the employee’s joining date is “2013-12-12”**

🡪Delete from employee where date of joining =”2013-12-12”;

------------------------------------------------------------------------------------------------

Select city, gender, sum(salary) as totalsalary, count(id) as (total employees)

From employee

Where gender=’male’-🡪To filter the rows we used where clause.

Group by city, gender.

(or) The other way of to get the same result instead of where clause we will use

Having clause.having clause should come after groupby

Select city, gender, sum(salary) as totalsalary, count(id) as (total employees)

From employee

Group by city ,gender

Having gender=’male’

Difference between where clause and having clause:

🡪where clause can be used with-select, insert, and update statements, whereas having clause can only be used with the select statement.

🡪where filters rows before aggregating (grouping), whereas, having filters groups, after the aggregations are performed.

🡪Aggregate functions cannot be used in the where clause, unless it is in a sub query contained in a having clause, whereas aggregate functions can be used in having clause.

* **Having Clause:**

🡪when we need to place any conditions on the table’s column, we use the where clause in SQL. But if we want to use any condition on a column in group by clause at that time, we will use the having clause with the group by clause for column conditions.

**Syntax:**

🡪Table name group By column Name having condition;

**Ex: waq to display the name of employees, salary, and city where the employees maximum salary is greater than 40000 and group the results by designation.**

Select name, city, Max(salary)

as salary

from employee

group by designation having max(salary)>40000;

Ex:2: w.a.q. to display the name of employees and designation where the sum of an employee’s salary is greater than 45000 and group the results by city.

Select name,designation,sum(salary)As salary

from employee

group by city having sum(salary)>45000;

* **Order By Clause:**

🡪It is used to sort the records in ascending or descending order.

**Syntax:**

**Select expressions**

**From tables**

**[Where conditions]**

**Order by expression [asc][desc]**

**Order by without using Asc/Desc attribute:**

**If we use sql order by clause without specifying the Asc and Desc modifier then by default you will get the result in ascending order.**

🡪Select\*

From officers

Where address=’lucknow’

order by officer\_name;

**orderby with ASC attribute:**

🡪select\* from officers where address=’lucknow’

order by officer-name ASC;

**orderby with Desc attribute:**

🡪select\* from officers where address=’lucknow’

order by officer-name DESC;

**using both Asc and Desc attributes:**

🡪select officer\_name ,address

from officers

where officer\_id<5

order by officer\_name Desc, address Asc;

* **Normalization**:

🡪Normalization is the process of organizing the data in the database.

🡪Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like insertion, update, and deletion anomalies.

🡪It divides the larger table into smaller and links them using relationships.

🡪The normalization is used to reduce redundancy from the database table.

**Redundancy**: Repetition of unnecessary data in a table is known as redundancy.

* **Anomalies**: The side effects that occur on table while performing dml operation on a table is turn as anomalies, such as insert, update, delete operations.
* Types of anomalies:

Insertion: Insertion anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.

Delete Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

Update: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

* **Why do we need Normalization?**

🡪The main reason for normalizing the relation is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

* **Types of Normal Forms:**
* **1NF:**

**🡪**Each & every cell should have a single value.

🡪As a record it should be unique.

* **2NF:**

🡪A relation will be in 2NF if it is in 1NF, and all non-key attributes are fully functional dependent on the primary key.

🡪If we come across any dependency, there table should be divided & linked with the relationship.

* **3NF:**

🡪A relation will be in 3NF if it is in 2NF, it does not have transitive functional dependency.

* **BCNF**: A stronger definition of 3NF is known as Boyce Codd’s normal form.
* **4NF**: A relation will be in 4NF if it is in Boyce Codd’s normal form and has no multi-valued dependency.
* **5NF**: A relation in 5NF.If it is in 4NF and does not contain any join dependency, joining should be lossless.
* **Advantages of Normalization**:

🡪Normalization helps to minimize data redundancy.

🡪Greater overall database organization

🡪Data consistency within Database.

🡪Much more flexible database design.

🡪Enforces the concept of relational integrity.

* **Disadvantages of Normalization:**

🡪You must start building the database before knowing what user needs.

🡪The performance degrades when normalizing the relations to higher normal forms i.e,4Nf,5Nf.

🡪It is very time-consuming and difficult to normalize relations of a higher degree.

🡪Careless decomposition may lead to a bad database design, leading to serious problems.

* **Types of Attributes**
* Key Attribute
* Composite Key
* Non-Composite key
* Key Attribute:

🡪The attribute which determines the other attribute is called as key attribute.

* **Composite Key attribute:**

**🡪**It is nothing but the combination of one or more attributes uniquely determines the set of other attributes.

* **Non composite KEY :**

🡪The attributes in a relation does not determine any other attribute is called as non-composite key attribute.

* **Functional Dependency:**

🡪It is nothing but a relationship which is used to determine another attribute within the relation.

* **Types of Functional Dependency:**

**Total Functional Dependency (TFD):**

🡪In a relation if all the attributes are depending on a single attribute (key attribute) is called Functional Dependency.

* **Partial Functional Dependency:**

🡪A relation is said to have partial functional dependency if it behaves the following characteristics.

* Rule1: The relation has composite key attribute.

A relation should have attribute which is depending on another attribute which is a part of the composite key attribute.

* **Transitive Functional Dependency:**

🡪Tfd is a relation if it behaves the following characteristics.

🡪All the attribute dependent on a key.

🡪There is an attribute relation which is dependent another attribute which is not a part of key attribute.

🡪It has minimum 3 columns should be there. It works on the condition from right to left, not to the left to right.

**Index:**

🡪An index is one of the important paths to make the performance of SQL server database high.

🡪It makes the querying process fast by providing easy access to rows in data tables, similar to how a books index will quickly locate information inside that book.

🡪If we don’t have an index, then it is very tough to locate the specific data type. We can have create index, drop index, alter index, SQL commands for creating new indexes, updating existing indexes, and deleting indexes in SQL server.

🡪AN index is a set of keys made up of single or multiple columns in a table or view. They stored in a structure(B\_tree) that helps SQL server users quickly and efficiently to find rows or rows associated with key values.

🡪There are mainly two types of indexes.

1.Clustered 2.Non Clustered

**1.Clustered**

🡪 clustered indexes use key values for sorting and storing data rows in tables or view.

🡪They are included in the index definition. It always stores the index values in a B-tree structure where the actual data is stored in leaf node.

🡪since the data rows are stored in one direction each table can only one single clustered index.

🡪A table stored the rows in sorted order only when the table has a clustered index, we can refer to a clustered table as one of that has a clustered index.

🡪If a table doesn’t have a clustered index, it rows are stored in heap, which is an unordered structure.

🡪The main benefit of a clustered index is the data is physically stored by the clustered key value in your storage system, and searching for a range of values will be fast.

🡪Its main disadvantage is the last page insert latch contention that inserts data only at the end of the clustered Index.

Non-Clustered:

* The structure of non-clustered index is similar to the clustered index except that the actual data is not contained in the leaf nodes. A non-clustered index has the non-clustered index key values, and each key-value entry contains a reference to the actual data.
* The main benefit of the non-clustered index is to speed up query performance. Its main disadvantage is the extra overhead needed to maintain the index during DMLoperations.

**Syntax: Create[unique|clustered\Nonclustered]index index-name On table\_name column\_name;**

If you want to create multiple index columns,

Create index inex\_name on table\_name(column1, column2..)

**Create an index:**

**Create index index\_age On student (age);**

* **Joins**

**🡪Joins in SQL server are used to retrieve the data from 2 or more related tables. (or) SQL allows developers to retrieve columns from multiple tables using joins**

**It is used to combine data or rows from two or more tables based on a common field between them.**

**In general tables are related to each other using foreign key constraints.**

**In SQL there are 3 types of Joins.**

**1.Inner Join or join**

**2.Outer join**

1. **Left outer join**
2. **Right outer join**

**3.Full join**

**4.Cross join or cartesian join**

Consider two tables:

StudentList:

Create table StudentList(

Roll-No int,

Name varchar(20),

Address varchar(20),

Phone int,

Age int

);



**StudentCourse:**

**Create table StudentCourse(**

**Course\_id int,**

**Roll\_No int**

**);**



* **Inner Join**
* Inner joins are the most common type of join and are the default join used by the SQL server.
* Inner Join keyword selects all rows from both the tables as long as condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e.values of the common field will be the same.
* When we use the = operator in the join condition, it is called an equi-join, since it shows results where tables are combined based on a common column.
* **Syntax: select table1.column1, table1.column2,table2.column1**

**From table1**

**Inner join table2**

**On table1.matching\_column=table2.matching\_column**

* Matching \_column: column common to both the tables.

Select StudentList.Name, StudentList.Address,StudentCourse.Course\_id

From StudentList

Inner join StudentCourse

On studentList.Roll\_no=StudentCourse.Roll\_No;

**Outer Join:**

* **Left Outer Join:**

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. If there is no matching row on the right side, the result will contain null. Left join is also known as left outer join.

* **Syntax: select table1.column1, table1.column2,table2.column1**

**From table1**

**Left join table2**

**On table1.matching\_column=table2.matching\_column**

Select Studentss.admission\_no,Studentss.first\_name, Studentss.last\_name,Fee.Course,Fee.amount\_paid

From Studentss

Left outer join Fee

On Studentss.admission\_no=Fee.admission\_no;

* **Right Outer Join:**

This Join returns all the rows of the table on the right side of the join and matches rows for the table on the left side of the join.If there is no matching

Rows on the left side, the result will contains null. Right Join Is also known as right outer join.

* **Syntax: select table1.column1, table1.column2,table2.column1**

**From table1**

**Right join table2**

**On table1.matching\_column=table2.matching\_column**

Select Studentss.admission\_no,Studentss.first\_name, Studentss.last\_name,Fee.Course,Fee.amount\_paid

From Studentss

Right outer join Fee

On Studentss.admission\_no=Fee.admission\_no;

**Full Outer Join:**

It creates the result-set by combining results of both left join and right join. The result set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain null values.

* **Syntax: select table1.column1, table1.column2,table2.column1**

**From table1**

**Full join table2**

**On table1.matching\_column=table2.matching\_column**

Select Studentss.admission\_no,Studentss.first\_name, Studentss.last\_name,Fee.Course,Fee.amount\_paid

From Studentss

Full join Fee

On Studentss.admission\_no=Fee.admission\_no;

* **Cross Join or cartesian Join:**

The cartesian join is also known as cross join.In a cartesian join there is a join for each row of one table to every row of another table.

This usually happens when the matching column or where- condition is not specified.

* **Syntax: select table1.column1, table1.column2,table2.column1**

**From table1**

**Cross join table2**

Select Studentss.admission\_no,Studentss.first\_name, Studentss.last\_name,Fee.Course,Fee.amount\_paid

From Studentss

Cross join Fee

* **Self Join:**

In self Join a table is joined to itself. That is each row of the table is joined itself. That is each row of the table is joined with itself and all other rows depending on some conditions. In other words we can say that it is a join between two copies of the same table.

* **Syntax:Select a. column1,b.column2**
* **From table-name.a,table-name.b**
* **Where some condition;**

Select Studentss.admission\_no.Fee.course

From Studentss,Fee

Where students.admission\_no=Fee.admission\_no

**Functions:**

* Functions are the reusable programs.
* To execute the functions we need to pass some inputs and those inputs sometimes called as parameters or arguments.

**Types of Functions:**

1.Built in Functions : In built in functions already functions are defined just by calling them we are using it.

2.User Defined Functions: In user defined functions we come across pl/sql.

**Types of Built in Functions:**

1.single row function: In S.R function single output for individual input.

2.Multi row function: In M.R function it returns single o/p for individual input.

**Types of Single row function**

1.Character Function

2.Number Function

3.General Function

4.Conversion Function

5. Date Function

**Types of Character Function:**

1.Case Manipulation Function

2. Character Manipulation Function

**Types of Case manipulation:**

1.**Upper**: It is used to convert all letters in a given string to uppercase.

Ex: Upper(string)

Select upper(‘divya’) from dual.

2.**Lower**: It is used to convert all letters in a given string to lowercase.

Ex: Lower(string)

Select lower(‘divya’) from dual.

3.**Initcap**: It is used to convert the first letter in a word to uppercase and rest of the letter into lowercase. Ex:select initcap(‘divya’) from dual;

1. **Character Manipulation:**

1.Concat: It is used to join multiple strings.

* Syntax: concat(Arg1,Arg2)
* Ex:. select concat(‘hi’,’hello’) from dual;
* Select concat(‘first’,’second’) from dual;
* Select contact(id,name,work-date) from employee-table;

**2.Length:**

It is used to find the length of the given string. If string is null then length o the function returns null and not zero.

If the string contains extra space at the start, or in between or at the end of the string, then the length of the function includes the extra space too and returns the complete length of the string.

* **Syntax**: select Ename, Length (Ename)
* From emp.
* Ex: select Len (‘Learning is Fun’) from dual; o/p:15
* Select Len (‘write an interview experience’) from dual; o/p:34.
* Select Len (‘’) from dual; or select length(null) from dual; o/p:--
* Write a query to display all the employee details whose name consists of exactly 5 characters:
* Select \* from employee where length (Ename)=5;

**3.Substr:**

**🡪**It is used to find or extract the position of the given substring in a string.

🡪This function returns a portion of a string from a given start point to end point .If a substring length is not given ,then substring returns all the character till the end of string(from the starting position specified).

* SELECT SUBSTRING('Database Management System', 9,7) FROM DUAL;---manage---
* select First\_Name,SUBSTRING(First\_Name,1,1) from Employee\_Grade;---first character in their name---
* select First\_City, SUBSTRING(First\_City,2,1) from Employee\_Grade;---first 2characters or second character--
* select Second\_City, SUBSTRING(Second\_City,-2,1) from Employee\_Grade;--last character---
* select \*
* from Employee\_Grade
* where SUBSTRING(Second\_City,2,1)='h';---whose name second character is a--
* select \*
* from Employee\_Grade
* where SUBSTRING(New\_City,-1,1)='w';

4.Instr:

🡪It is used to find the position of the given substring. If substring is not present, then by default it returns 0.

* SELECT INSTR('Google apps are great applications','app',1,2) FROM DUAL;
* Select instring (‘javatpoint’,’p’) as instr\_p\_Position;-----it gives 6 as o/p---

**5.Reverse:**

🡪The reverse string function in SQL returns the string in the reverse order. It shows the last character of the string at the first position and the first character at the last position in the output of the query.

* EX: select Reverse(Original\_string);
* Select reverse(‘javatpoint’);---o/p: --tniopavaj;
* Select Reverse (‘new delhi is the capital of India’)—o/p:’aidni fo latipac eht si ihled wen’.
* Select reverse (‘578442297425’);---o/p: 524792244875.
* Select New\_City, Reverse (New\_City) as revesre\_NewCity FROM Employee Grade;
* Select First\_City, Reverse (New\_City) as revesre\_NewCity FROM Employee Grade.
* Select First\_City, Reverse (First\_City) as revesre\_FirstCity FROM Employee Grade.

**6.Replace:**

🡪It is used to replace set of characters by another sequence of characters.

* Select last\_name, replace(last\_Name,’a’,’r’) as replace\_a\_r from Employee\_Grade;
* Select New\_City,replace(New\_City,’a’,’h’) as replace\_a\_h from Employee\_Grade;
* Select Work\_remarks, replace (Work\_remarks,82,85) as replace\_82\_85from Employee\_Grade;
* Select Grade, replace (Grade,’A2’,’A1’) as replace\_A2\_A1 from Employee\_Grade;

**7.Trim:**

🡪Trim function is used to remove some specify character in given string.

Types of Trim:

1.LTrim: It is used to move some specify characters and space in a given string from left hand side.

Select LTRIM(‘javatpoint’,’geeks’);

Select LTRIM (‘new Delhi is the capital of india’,’new delhi is the’);

Select LTrim(‘###98221545##’,#’);

Select LTrim(‘202120212021javaTpoint’,’2021’);

Select LTrim(‘90287javaTpoint’,’0123456789’);

2.RTrim: It is used to remove some specify characters and space in a given string from right side.

* Select rtrim(‘new delhi is the capital of india’);
* Select rtrim(‘###98221545##’,’#”);
* Select rtrim(‘2021javaTpoint2021’,’2021’)

3.Trim: (Both): The trim() is a string function of MySQL. It removes the blank spaces from the head(leading) and tail(trailing) of the given string.

* **Trim ([leading | Trailing | Both] [ trim character from] string)**
* Select trim (leading ‘mysql’ from ‘mysql\_javatpoint’);
* Select trim (both ‘mysql’ from ‘mysql\_javatpoint\_mysql);
* Select trim (trailing ‘mysql’ from ‘mysql\_javatpoint\_mysql);

**Types of Number Functions:**

1.Sqrt: It is most commonly used function. It takes any numeric values and returns the square root value of that number.

Select sqrt(value)

* Ex: select Sqrt (100);

Select sqrt(36) as result1, sqrt(10.16) as result2;

2.Pi: This function is used to get the constant value of pi, which is approximately equal to 3.14

* Select Pi()

3.Square: It is used to find square of any number.

* Select square(value);
* Select square(25);

4.Round : It returns a round a value to the nearest specified decimal place.

* Select round(value, number of decimals)
* Select round(125.35,1);
* Select round(125.35,2);

5.Ceiling:It is used to find the next highest value(integer)

Select ceiling(value)

* Ex: select ceiling(45.56)

6.Floor: Floor() function returns the next lowest value.

* Select Floor(value)
* Select floor(45.56).

**Types of General :**

1.NVL (null value logic1):

🡪NVL() converts a null values to an actual value. Data types that can be used are date, character, and number. Data type must match with each other i.e. expr1 and expr2 must of same data type.

* Syntax: NVL (expr1,expr2)
* If Expression1 is null then exp2 will be return.
* If expression1 is not null then exp1 will be return.
* Note : while executing a query instead of NVL we can use ISNULL .
* Select isnull(15,20) from dual or select isnull(15,20) as result from dual;
* Select isnull(null,20) from dual
* Select isnull(1.027584,1.0275384) from dual
* Select isnull(null,1,987650) from dual
* Select isnull(‘James’,’Anderson’) from dual.

🡪Expr1 is the source value or expression that may contain a null.

🡪 Expr2 is the target value for converting the null.

2. NVL2

3.coalesce 4.nullif 5.lnnvl 6.nanvl

**Types of Conversion:**

When you define expressions and local variables then you should specify what type of data will be sorted in those objects such as text, data, money, dates, numbers, or characters.

🡪strings data types such as char and varchar.

Decimal values such as float and real.

🡪Binary strings such as binary.

🡪Date and time data types such as date, time, timestamp, datetime.

🡪Numeric data types such as int, double and big int.

On the basis of this there are two types of conversions in the data first implicit types conversion and the second is explicit data type conversion.

🡪In implicit conversion server can automatically convert the data from one to another (i.e., varchar to char and int to float) but in explicit data types conversion it can be done by the user side).

Implicit data-Type Conversion: In this type of conversion the data is converted from one type to another implicitly (by itself/automatically).

Create table Employes(

Employe\_id int primary Key,

First\_name varchar(30),

Salary int);

Insert into employes(employe\_id,first\_name,salary) values(100,’steven’,24000)

Insert into employes(employe\_id,first\_name,salary) values(101,Neena ,17000)

Insert into employes(employe\_id,first\_name,salary) values(102,Lex ,17000)

Insert into employes(employe\_id,first\_name,salary) values(103,John ,11000)

Insert into employes(employe\_id,first\_name,salary) values(104,Robert ,12000)

Insert into employes(employe\_id,first\_name,salary) values(105,leo ,10000)

Select employe\_id,first\_name,salary from employes where salary>15000;

1.To-Number 2. To-Char 3. To -Date

**Types of Date Function: or Aggregate functions**

1.Max 2. Min 3.Avg 4.Count 5.Sum.

----------------------------------------------------------------------------------------------------------

* **Views:**

🡪views are kind of virtual tables. A view also have rows and columns as they are in a real table in the database.

🡪we can create a view by selecting fields from one or more tables present in the database.

🡪A view can either have all the rows or specified rows based on the certain condition.

Create table StudentDetails(

s-id int,

name varchar(20),

address varchar(30)

);

* Insert into StudentDetails(s-id,name,address) values(1.’harsh’,’kolkata’ );
* Insert into StudentDetails(s-id,name,address) values( 2,’ashish’,durgapur’);
* Insert into StudentDetails(s-id,name,address) values( 3,’Pratik’,’delhi);
* Insert into StudentDetails(s-id,name,address) values( 4’,Dhanraj’,’bihar’);
* Insert into StudentDetails(s-id,name,address) values(5,’ram’,’rajasthan’ );

Create StudentMarks(id int,

Name varchar(20),

Marks varchar(20),

Age int);

Insert into StudentMarks(id,name,marks,age) values(1,’harsh’,90,19);

Insert into StudentMarks(id,name,marks,age) values(2,’suresh’ ,50,19);

Insert into StudentMarks(id,name,marks,age) values(3,’pratik’ ,85,19);

Insert into StudentMarks(id,name,marks,age) values(4,’dhanraj’ ,95,21);

Insert into StudentMarks(id,name,marks,age) values(5,’ram’ ,85,18);

**Syntax:**

**Create view view-name as**

**Select column1,coulmn2..**

**From table-name**

**Where condition;**

**Creating a view from a single table**

**Create view vwdetailsview as**

**Select name,address**

**From StudentDetails**

**Where s\_id<5;**

**Select \* from vwdetailsview**

**Create view course\_enrolled**

**As**

**Select first\_name, last\_name, course, amount\_paid**

**From Studentss as S**

**Inner join fee F**

**On S.admission\_no=F.admission\_no;**

**Select \* from course-enrolled;**

**Rename:**

**Sp\_rename view\_old-name, view\_new-name**

**Ex:sp\_course\_enrolled,coursevw**

**------------------------------------------------------------**

**Update**

**Alter view course\_enrolled**

**As**

**Select first\_name, last\_name, course, city,amount\_paid**

**From Studentss as S**

**Inner join fee as F**

**On S.admission\_no=F.admission\_no;**

**Drop**

**Drop view [if exists] schema\_name.view\_name;**

**Drop view course\_enrolled.**

* **Triggers:**

🡪A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs.

🡪For Example a trigger can be invoked when a rows is inserted into a specified table or when specific table columns are updated in simple word a trigger is a collection of sql statements with particular names that are stored in system memory.

🡪It belongs to a specific class of stored procedure that are automatically invoked in response to database server events.

Every trigger has a table attached to it.

🡪The following are the main characteristics that distinguish triggers from stored procedures.

* We cannot manually execute/invoked triggers.
* Triggers have no chance of receiving parameters.
* A transaction cannot be committed or rolled back inside a trigger.

Syntax:

We can create a trigger in sql server by using the create trigger

Create trigger schema.trigger\_name

On table-name

After { insert,update,delete}

[not for replication]

As

{sql\_statements}

When we use Triggers?

Triggers will be helpful when we need to execute some events automatically on certain desirable scenarios.

If the primary table made any changes in such scenarios we could create a trigger to insert the desired data into a separate table.

**Create table employeeTr(**

**Id int primary key,**

**Name varchar(45),**

**Salary int,**

**Gender varchar(12),**

**Departmentid int**

**)**

**Create table employee\_Audit\_Test**

**(**

**Id int identity,**

**Audit\_Action text**

**)**

**-------------------------------------------------------------------------**

**Create trigger trinsertEmployee**

**On EmployeeTr**

**For inser**

**As**

**Begin**

**Declare @id int**

**Select @id=id from inserted**

**Insert into employee\_Audit\_test**

**Values(‘new employee with id=’+cast(@id as varchar(10))+ ‘is added at’+ cast(getdate() as varchar(22)))**

**End**

**After creating a trigger , we will try to add following records**

**Insert into EmployeeTr values(6,’peter’,62000,’male’,3)**

**Select \* from employee-audit-test**

**--------------------------------------------------------------------------------------------------**

**Create trigger trinsertEmployee**

**On EmployeeTr**

**For inser**

**As**

**Begin**

**Declare @id int**

**Select @id=id from deleted**

**Insert into employee\_Audit\_test**

**Values(‘new employee with id=’+cast(@id as varchar(10))+ ‘is added at’+ cast(getdate() as varchar(22)))**

**End**

**Select \* from employee-audit-test.**

**Delete from Employee where id=2;**

**Triggers are 3 types.**

1.DMl triggers: These are automatically fired when an insert, update, or delete event occurs on a table.

2.DDL are automatically invoked when a create, alter, drop event occurs in database scoped event.

3.Logon: it is invoked when a logon event is raised when a user session is established.

**ER(Entity Relationship) Diagram**

* Er model stands for an Entity relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
* It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
* In Er modelling, the database structure is portrayed as a diagram called an entity relationship diagram.



**Entity:**

* An Entity may be any object ,class, person, or place. In the ER diagram an entity can be represented as rectangles.

Consider an organization as an example manager ,product, employee,department etc can be taken as an entity.



**A.Weak Entity:**

An entity that depends on another entity called a weak entity. The weak entity doesn’t contain any key attribute of its own. The weak entity is represented by a double rectangle



**2.Attribute:**

The attribute is used to describe the property of an entity.Eclipse is used to represent an attribute.

Ex; id,age,contact number,name,etc can be attributes of a student.



**A.Key attribute**

The key attribute is used to represent the main characteristics of an entity.It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



**B.Composite Attribute:**

An attribute that composed of many other attributes is known as composite attribute.

The composite attribute is represented by an ellipse and those ellipses are connected with ellipse.



**C:Multivalued attribute:**

An attribute can have more than one values. These attributes are known as a multivalued attribute .The double oval is used to represent multivalued attribute.

Ex: A student can have more than one phone number.



**d.Derived Attribute:**

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

Ex: A person’s age changes over time and can be derived from another attribute like Date of birth



**3.Relationship:**

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



**Types of relationship are as follows:**

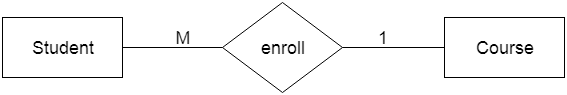
* When only one instance of an entity is associated with the relationship, then it is known as one-to-one relationship.

Ex: A female can marry to one male and a male can marry to one female.



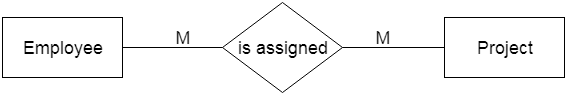
* b.one -to-many relationship
* When only one instance of the entity on the left, and more than one instance of an entity on the right associate with the relationship then this is known as a one-to-many relationship.
* Ex: scientist can invent many inventions, but the invention is done by the only specific scientist.
* **C.Many-to-one relationship**
* When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

Ex: student enrolls for only one course, but a course can have many students.



* **d.Many-to-many relationship**
* when more than one instance of the entity on the left,and more than one instance of an enity on the right associates with the relationship then it is known as a many-to-many-relationship.

Ex: employee can assign by many projects and project can have many employees.



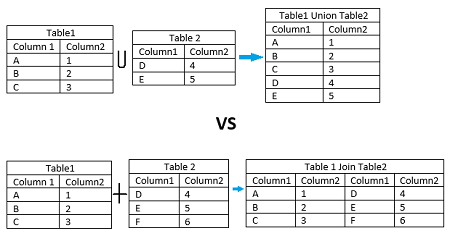
**Difference between Union and union all**

**Union:**

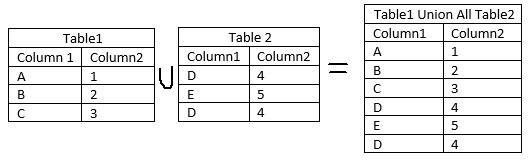
* Union operator in Mysql allows us to combine two or more results from multiple select queries into a single result set.
* It has a default feature to remove the duplicate rows from the tables.
* This Operator syntax always uses the columns name in the first select statement to be the column names of the output.
* Mysql union must follow these basic rules:
* The number and order of the columns should be the same in all queries.
* The corresponding columns position of each select query must have a compatible data type.
* The column name selected in the different select queries must be in the same order.
* The column name of the first select query will be the column names of the output.

Note:

* We must have to know that union and join are not same.
* Join combines data from multiple different tables, whereas union combines data from multiple similar tables.
* Join appends the output horizontally, whereas union combines the result set vertically.
* The below visual representation explains it more clearly:



* **Union All**
* The Union All operator combines two or more results from multiple select queries and returns all records into a single unit set. It does not remove the duplicate rows from the output of the select statements.
* We can understand it with the following visual representation:



Examples: create table Student1(

Stud\_id int not null,

Name varchar(20),

Email nvarchar(30),

City varchar(20),

);

Insert into Student1(Stud\_id, Name, Email,City) values(1,’peter’,’peter@javatpoint.com’,’texas’);

Insert into Student1(Stud\_id, Name, Email,City) values(2,’suzi’,’suzi@javatpoint.com’,’california’)

Insert into Student1(Stud\_id, Name, Email,City) values(3,’joseph’ ,’joseph@javatpoint.com’,’alaska’);

Insert into Student1(Stud\_id, Name, Email,City) values(4,’andrew’ ,’andrew@javatpoint.com’,’losangeles’);

Insert into Student1(Stud\_id, Name, Email,City) values(5,’Brayan’ ,’brayan@javatpoint.com’,’newyork’);

Ex:Create table student2(

Stud\_id int not null,

Name varchar(20),

Email nvarchar(30),

City varchar(20)

);

Insert into Student1(Stud\_id, Name, Email,City) values(1,’stephen’ ,’stephen@javatpoint.com’,’texas’);

Insert into Student1(Stud\_id, Name, Email,City) values(2,’joseph’,’joseph@javatpoint.com’,’losangeles’);

Insert into Student1(Stud\_id, Name, Email,City) values(3,’peter’,’peter@javatpoint.com’,’california’);

Insert into Student1(Stud\_id, Name, Email,City) values(4,’david’,’david@javatpoint.com’,’newyork’);

Insert into Student1(Stud\_id, Name, Email,City) values(5,’maddy’,’maddy@javatpoint.com’,’losangeles’);

Select City from student1

Union

Select city from student2

Order by city;

Select City from student1

Union all

Select city from student2

Order by city;

Differences between Union and union all

|  |  |
| --- | --- |
| Union | Union All |
| It combines the result set from multiple tables and returns distinct records into a single result set. | It combines the result set from multiple tables and returns all records into a single result set. |
| Syntax: select column\_list from table1  Union  Select column\_list from table2; | Syntax: select column\_list from table1  Union all  Select column\_list from table2; |
| It has a default feature to eliminate the duplicate rows from the output. | It has no feature to eliminate the duplicate rows from the output. |
| Its performance is slow because it takes time to find and then remove duplicate records. | Its performance is fast because it does not eliminate the duplicate rows. |
| Most database users prefer to use this operator. | Most database users do not prefer to use this operator. |

* **Denormalization:**

It is a database optimization technique in which we add redundant data to one or more tables.

* **Super Key:**

A super key is a set of attributes of a relation schema upon which all attributes of the schema are functionally dependent. No two rows can have the same value of super key attributes.

* **Candidate Key:**

A candidate key is a minimal super key, i.e.no proper subset of candidate key attributes can be a super key.

**SQL sub query:**

A sub query is a within another SQL query and embedded within the where clause.

**Important Rules:**

A subquery can be placed in a number of SQL clauses like where clause, from clause, having clause.

A subquery is a query within another query. The outer query is known as the main query, and the inner is known as a subquery.

Sub queries are on the right side of the comparison operator.

A subquery is enclosed in parentheses.

The inner query will execute first.

In the subquery, order by command cannot be used. But group by command can be used to perform the same function as order by command.

* Syntax for subquery with select statement:

**Select column\_name**

**From table\_name**

**Where column\_name expression operator**

**(select column\_name from table\_name where…..);**

Consider the employee table have the following records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Age | Address | salary |
| 1 | John | 20 | us | 2000 |
| 2 | Stephen | 26 | Dubai | 1500 |
| 3 | David | 27 | Bangkok | 2000 |
| 4 | Alina | 29 | UK | 6500 |
| 5 | Kathrin | 34 | Bangalore | 8500 |
| 6 | harry | 42 | China | 4500 |
| 7 | Jackson | 25 | Mizoram | 10000 |

Select \*

From Employee

Where Id in(select id from employee where salary>4500);

* **Subquery with insert statement**

**Syntax:**

**Insert into table \_name(column1,column2,column3…)**

**Select\* from table\_name**

**Where value operator.ar**

Consider a table Employee\_bkp with similar as employee

Now use the following syntax to copy the complete employee table into the employee\_bkp table.

Insert into employee\_bkp

Select \*from Employee

Where id in (select id from employee);

* Subquery with update

**Update table**

**Set column\_name=new value**

**Where value operator**

**(select column\_name**

**From table\_name**

**Where condition);**

Ex: update Employee

Set salary=salary\*0.25

Where age in ( select age from customers \_bkp

Where age>=29);

* **Subquery with delete table**

**Delete from table\_name**

**Where value operator**

**(select column\_name**

**From table\_name**

**Where condition);**

Delete from employee

Where age in (select age from employee-bkp

Where age>=29);