SQL : Structured Query language

Used to store, update, maintain and delete the data from database. These are also called as **CRUD – Create Read Update Delete.**

1. There are two types of DB :
    a. *Relation DB (Parent child DB)*
    b. *No Relational DB (Key value pairs)*
2. In real time we will have a lakhs of rows will be there inside a DB – when we are retrieving the data it will take time (the no.of rows are directly proportional to time taken)
3. To insert data automatically we use the API's – we are having triggers for this.
4. The main disadvantage of using the relational DB > when the data is increasing parallel we have to increase the system resource > this is called as horizontal scaling.(there is also having a limitation for it)
5. When there is a multiple request for a single table then the system resource can't handle all of them. Hence it will lead to server down
6. Generally this RDB having limitation to horizontal and vertical scaling
7. Whereas the NRDB will support the horizontal and vertical scaling (horizontal – resources, vertical – number of servers).
8. In NRDB we will store the values as Key value pairs
9. NRDB is having more scalable than RDB.
10. DB – software | DBMS – software to manage DB

RDB :

There are many DBMS where we can learn this SQL – MySQL, Oracle,…

Min Req :

we need the MySQL server : https://dev.mysql.com/downloads/installer/

We need the MySQL WorkBench : https://dev.mysql.com/downloads/workbench/

We need the MS Visual C++ 2013 Redistribution file to install the server, if you are facing issue with the MySql server download :

https://download.microsoft.com/download/2/E/6/2E61CFA4-993B-4DD4-91DA-3737CD5CD6E3/vcredist_x86.exe

Note : If are not able to install the MySql then please install the server 1st separately.

Note : No need to set the path unlike java/ python.

We can communicate with the MySQL using the below ways

1. MySql client
    a. It is a cmd prompt where we can execute the queries.
2. MySql Yog
3. MySql Workbench → GUI apps

We will having the following **Datatypes** in SQL :

1. Numbers
    a. Int
    b. Bigint
    c. Decimal – we can give two numbers (M, N).
        a. M – Total no.of digits before dot(.)
        b. N – Total no.of digits after dot(.)
    d. Float
    e. Double
2. String
    a. Char
    b. Varchar
    c. Text
3. Boolean
4. Binary
    a. Blob
    b. Varbinary
5. Datetime
    a. Date (yyyy – mm- dd)
    b. Time
    c. Datetime
    d. Timestamp

Comments in DB :

We can use the # or -- to add the comment.

In SQL the commands are classified into 5 types

1. **DDL – Data Definition Lang**
    a. Create
    b. Alter
    c. rename
    d. Truncate
    e. Drop
2. **DML – Data Manipulation**
    a. Insert
    b. Update
    c. Delete
3. **DQL – Data Query Language**
    a. Select
4. **DCL – Data Control Language**
    a. Grant
    b. Revoke
5. **TCL – Transaction Control Language.**
    a. Commit
    b. Rollback

1.DDL :

The structure related things will fall under DDL.

1. **Create** : We can create table using the create

a. Syntax for Database Creation :

CREATE DATABASE DBName;

Example :

create database Demo;

b. Syntax for Table Creation :

CREATE TABLE tableName (colName1 dataType(size), colName2 DataType,….n);

Example :

create table test(id int(2), name varchar(10));

2. **Alter** : We can add, delete, rename or modify columns in an existing table.

a. Syntax to add a new column :

ALTER TABLE table_Name ADD col_name datatype;

Example :

ALTER TABLE test ADD email VARCHAR(20);

b. Syntax to delete a existing column :

ALTER TABLE table_Name DROP COLUMN Column_name;

Example :

ALTER TABLE test DROP COLUMN email;

c. Syntax to Modify the columnName and datatype :

ALTER TABLE table_Name MODIFY COLUMN column_Name datatype;

Example :

ALTER TABLE test MODIFY COLUMN email varchar(10);

d. Syntax to insert data at a specific position :

After :

ALTER TABLE table_Name modify column colName dataType AFTER colName;

Example :

alter table test modify column salary INT after id;

First :

ALTER TABLE table_Name modify column targetCol dataType First;

Example :

Alter table test modify column salary first;

3. **Truncate** : Will delete the data but the structure will remain.

a. Syntax to delete the entire data inside the table :

TRUNCATE TABLE table_Name;

4. **Drop** : Will delete the entire table along with the data.

a. Syntax to drop the table :

DROP TABLE tableName;

Example :

drop table test;

5. Rename :

Used to rename the table in a database.

Syntax :

Rename table tableName to newTableName;

2. **DML** :

It is used to manipulate the data.

1. ***Insert*** : used to insert data into table.
   a. ***Syntax*** :

   INSERT INTO Table_Name (colName) VALUES (col_Values);

   Example :

   insert into test values(1, 'Test1');

   b. ***Syntax*** :
   INSERT INTO table_Name (col1, col2) VALUES (val1, val2);
   Example :

   INSERT INTO test (id, name) VALUES (1, 'MIke');
2. ***Update*** : used to update the existing data.
   a. ***Syntax*** :

   UPDATE table_Name SET col_Name = value; // apply to entire column

   Example :

   alter table test add (times TIME, DOJ DATE);

   update test set times = CURRENT_TIME();

   update test set DOJ = current_date();

   select * from test;

   Output :

id	name	salary	times	DOJ
1	parker	50000	23:23:20	2024-03-10
4	raju	5000	23:23:20	2024-03-10
5	peter	3000	23:23:20	2024-03-10
6	rose	10000	23:23:20	2024-03-10
7	raju	5000	23:23:20	2024-03-10
8	peter	3000	23:23:20	2024-03-10
10	name	NULL	23:23:20	2024-03-10
NULL	NULL	NULL	NULL	NULL

   where condition; // when you want to use some condition

Condition :

In order to build a condition we use the conditional ops.s

NOTE : To update it is better to use the WHERE condition to update the data, as the above one will replace all the values inside the column

Note : If You are getting any error like where will be need to update use the below command.

```
SET sql_safe_updates=0;
```

3. ***Delete*** : used to delete the data inside the table.
   a. ***Syntax to delete a specific data***:

      DELETE FROM table_name where condition;

   Example :

      Delete from test where id = 9;

   b. Syntax to delete entire data :

      DELETE FROM table_name;

   Example :

      Delete from test;

3.DQL :

1. ***Select*** : used to display the data in a database.
   a. ***Syntax to show all the table data***:

      SELECT * FROM table_name;

   Example :

      select * from test;

   Output :

	id	name
▶	1	Test1

   b. Syntax to show specific data :

      SELECT colName / * FROM table_name WHERE Condition;

   Example :

      Select name, salary from test;

   c. Syntax to change the colName :

We can alter the display of the colName during output with the help of AS key word.

Syntax :

colName as displayName;

Example :

Select name AS FullName from test;

d. Using dot :

We can be more specific which column it needs to select.

Syntax :

tableName.colName;

Example :

test.name;

4. DCL :

1. ***Grant*** : the admin will grant the permissions to the user :
   a. Read
   b. Write
   c. Read & Write
2. ***Revoke*** : used to remove the given permissions from user

5.TCL :

Used to control the transactions. We are having the ACID properties in TCL.

1. ***Commit*** : Used to save the latest changes or save points

   Syntax :

   Commit;

   Example :

   Commit;
2. ***Rollback*** : used to roll back to the last save point

   Syntax :

   Rollback;

   Example :

   Rollback;

NOTE : We can use the describe table to get the what type of data we are storing in it.

Syntax :

Describe tableName;

Example :

Describe test;

Output :

	Field	Type	Null	Key	Default	Extra
	id	int	YES		NULL	
▶	name	varchar(10)	YES		NULL	

Constraints :

Constraints are some rules and regulations that we define to the columns when inserting the data into the table.

1. **NOT NULL** : It is used to make user to make sure that there are no Null values in it. It marks the column as default column.

   Syntax :

   colName datatype NOT NULL;

   Example :

   id int NOT NULL;

2. **Unique** : It will make sure that the data we are entering into the column is unique and it will not allow duplicates.

   Syntax :

   colName dataType UNIQUE;

   Example :

   Email varchar(20) UNIQUE;

3. **Primary Key** : It is the combination of NOT NULL + UNIQUE keys.

   Syntax :

   colName dataType PRIMARY;

   Example :

   Id int PRIMARY;

4. **Default** : It will add a default value to the column when user didn't provided any data.

   Syntax :

colName dataType DEFAULT ‘value’;

Example :

Email varchar(10) DEFAULT ‘test@mail.com’;

5. **Auto increment** : It is used to increase value by itself whenever we are adding data to the table.

Syntax :

colName dataType AUTO_INCREMENT;

Example :

Id int AUTO_INCREMENT;

NOTE : By default auto increment will consider the deleted data as one belongs to the table hence it will continue auto update from there.

Tip : To rectify that one we use the alter and reset the auto_increment to 1

Syntax :

Alter table table_Name auto_increment = value;

Example :

alter table test auto_increment = 1;

6. **Check** : We use the check to identify the info and insert the data into the table only if the condition is met to that column.

Syntax :

Check(condition);

Syntax to identify the check :

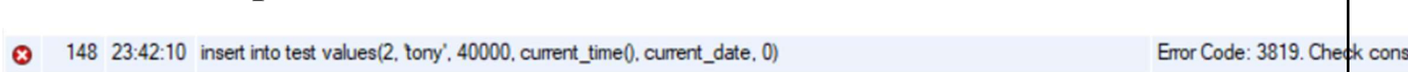
Constraint identifier check(condition);

Example :

alter table test add constraint inhand_sal_check check(inhand_sal > 0);

insert into test values(2, 'tony', 40000, current_time(), current_date, 0);

Output :



7. **Foreign keys** : To form a relation between the two tables we use the foreign key. The foreign Is the primary key of the other table.

Syntax :

foreign key (tableColName) references parent(colName);

Example :

foreign key (product_id) references product(id)

To delete Foreign key we use the alter query.

Syntax :

Alter table tableName drop foreign key foreign_key;

To rename the foreign key.

Syntax :

Alter table tableName add constraint rename foreign key (tableColName) references parent(colName);

Operators :

SQL uses a variety of operators for performing operations such as comparison, arithmetic, logical, and string manipulation. Here's an overview of some commonly used operators in SQL:

1. **Comparison Operators:**

- `=` : Equal to :

Will check if the value is equal to the given value or not.

Syntax :

colName = value;

Example :

id = 2

- `<>` or `!=` : Not equal to :

Will check if the value is not equal to the given value or not.

Syntax :

colName != value;

colName <> value;

Example :

Id != 2;

Id <> 2

- `<` : Less than :

Will check if the value is less than the given value or not.

Syntax ;

colName < value;

Example :

Id < 3;

- `>` : **Greater than :**

Will check if the values is greater than the given value or not.

Syntax :

colName > value;

Example :

Id > 3;

- `<=` : **Less than or equal to :**

Will check if the values are less than or equal to the given value or not.

Syntax ;

colName <= value;

Example :

Id <= 4;

- `>=` : **Greater than or equal to :**

Will check if the values are greater than or equal to the given value or not.

Syntax ;

colName >= value;

Example :

Id >= 4;

2. **Arithmetic Operators:**

- `+` *: Addition*
- `-` *: Subtraction*
- `*` *: Multiplication*
- `/` *: Division*
- `%` *: Modulo (remainder)*

3. **Logical Operators**:

id	name	sales	highest_sales
1	mark	200	1000
2	mike	300	2000
3	sofie	400	1000

- `AND` : Logical AND

Used to check the both conditions are true then only it will satisfy the condition.

Syntax :

Condition1 And condition2;

Example :

SELECT * from products where sales > 300 and highest_sales > 500;

Ouput :

id	name	sales	highest_sales
3	sofie	400	1000

- `OR` : Logical OR :

Used to check anyone of the given condition is true or not. If true will run the query or else will not.

Syntax :

Condition1 or condition2;

Example :

SELECT * from products where sales > 300 or highest_sales > 500;

Output :

id	name	sales	highest_sales
1	mark	200	1000
2	mike	300	2000
3	sofie	400	1000

- `NOT` : Logical NOT

Will only work on one condition and select if the condition is not met.

Syntax :

NOT condition

Example :

SELECT * from products where NOT sales >= 400;

Output :

id	name	sales	highest_sales
1	mark	200	1000
2	mike	300	2000

4. **Concatenation Operator:**

- `||` : Concatenates two strings (syntax may vary depending on the SQL dialect)

5. **Pattern Matching Operators:**

- **`LIKE`** *: Matches a pattern*

Wild cards :

In MySQL, wildcards are special characters used in conjunction with the `LIKE` operator in `SELECT` statements to perform pattern matching in string comparisons. They allow you to search for data based on patterns rather than exact matches. Here are the commonly used wildcards in MySQL:

1. **Percent sign (%) wildcard:** It matches any sequence of characters (including zero characters).

Syntax :

Where colName LIKE ‘letter%’;

- Example:

SELECT * from products where name like "m%";

Output :

id	name	sales	highest_sales
1	mark	200	1000
2	mike	300	2000

2. **Underscore (_) wildcard:** It matches any single character.

Syntax :

Where colName LIKE ‘character_’;

- Example:

SELECT * from products where name like "mi_e

Output :

id	name	sales	highest_sales
2	mike	300	2000

- `**IN**` *: Specifies multiple values :*

Syntax :

Select selector from tableName where col IN (val1..valn);

Example :

SELECT * from products where highest_sales IN (1000, 2000);

Output :

id	name	sales	highest_sales
1	mark	200	1000
2	mike	300	2000
3	sofie	400	1000

- `**BETWEEN**` : Specifies a range of values

Used specially in a single column

Syntax :

Between value1 AND value 2;

Example :

SELECT * from products where highest_sales between 1500 and 5000;

Output :

id	name	sales	highest_sales
2	mike	300	2000

6. **Null Comparison Operators:**

- `**IS NULL**` : Checks for NULL values

Syntax :

colName IS NULL;

Example :

select * from test where salary IS NULL;

- **`IS NOT NULL`** : Checks for non-NULL values

Syntax :

colName IS NOT NULL

Example :

select * from test where salary IS NOT NULL;

7. **Assignment Operator:**

- `=` : Assigns a value to a variable or column

8. **Set Operators (used in conjunction with `SELECT` statements):**

manager_id	manager_name	manager_salary	branch
1	Venu	20000.00	HYD
2	Banu	30000.00	SEC
3	Srinu	10000.00	WAR

emp_id	emp_name	emp_salary	emp_manager	branch
1	Raju	10000.00	2	SEC
2	Rani	20000.00	1	HYD
3	Mani	40000.00	3	WAR
4	Tina	5000.00	1	HYD
5	Shena	500.00	2	SEC
6	Vicky	50000.00	3	WAR
7	Raju	10000.00	2	SEC
8	Rani	20000.00	1	HYD
9	Mani	40000.00	3	WAR
10	Venu	5000.00	1	HYD
11	Shena	500.00	2	SEC
12	Vicky	50000.00	3	WAR

- **`UNION`** : Combines result sets and removes duplicates

Syntax :

Query1 union Query2;

Example :

SELECT * from managers UNION SELECT * from employee;

manager_id	manager_name	manager_salary	branch	experience
1	Venu	20000.00	HYD	NULL
2	Banu	30000.00	SEC	NULL
3	Srinu	10000.00	WAR	NULL
1	Raju	10000.00	2	SEC
2	Rani	20000.00	1	HYD
3	Mani	40000.00	3	WAR
4	Tina	5000.00	1	HYD
5	Shena	500.00	2	SEC
6	Vicky	50000.00	3	WAR

- **`UNION ALL` :** Combines result sets including duplicates

Syntax :

SELECT colName from table1 UNION ALL SELECT colName from table2;

Example :

SELECT manager_name as Emp_names from managers UNION ALL SELECT emp_name from employee;

Output :

Emp_names
Venu
Banu
Srinu
Raju
Rani
Mani
Tina
Shena
Vicky

- `INTERSECT` : Returns common rows between two result sets

Syntax :

SELECT colname from table1 INTERSECT SELECT colname from table2;

Example :

SELECT manager_name from managers intersect SELECT emp_name from employee;

Output :

manager_name
Venu

- `EXCEPT` or `MINUS` : Returns rows that are in the first result set but not in the second

Order by :

Used to sort the data into the database using a condition.

id	name	sales	highest_sales
1	mark	200	500
2	mike	300	1000
1	mark	200	500
2	mike	300	1000

Syntax ;

Select selector from tableName ORDER BY condition;

1. ASC :

We can display the data in ascending order.

2. DESC :

We can display the data in descending order

Example :

select * from products ORDER BY sales;

id	name	sales	highest_sales
1	mark	200	500
1	mark	200	500
2	mike	300	1000
2	mike	300	1000

Example for desc :

select * from products ORDER BY sales desc;

id	name	sales	highest_sales
2	mike	300	1000
2	mike	300	1000
1	mark	200	500
1	mark	200	500

We can provide two values to the order by instead of one as it will give priority to the second if the 1st col has the same info.

Syntax :

Select selector from tableName order by salary, id;

Example :

select * from test order by salary,id asc;

Limit ;

We can limit the no of row to be display with the help of limit.

Syntax :

Select selector from tableName limit no.of rows;

Example :

select * from products LIMIT 1;

id	name	sales	highest_sales
1	mark	200	500

We can limit the no.of rows from the desired row.

Syntax :

Select selector from tableName limit desiredRow, limit;

Example :

select * from products LIMIT 2,1;

Output :

id	name	sales	highest_sales
3	sofie	400	4000

NOTE : If the table is having less rows then the limit it will simple ignore the rest

To get the date and time :

To get Current Date :

To get the current date we use the current_data() method.

Syntax :

Current_date();

To get Current Time :

To get the current time we use the current_time() method.

Syntax :

Current_time();

To get the current date and time :

To get the current date and time we use the Now() method.

Syntax :

NOW()

Example :

create table timeExample(times time, dates date, Today datetime);

insert into timeExample values (Current_time(), current_date(), NOW());

select * from timeExample;

Output :

	times	dates	Today
▶	23:26:44	2024-03-10	2024-03-10 23:26:44

Note : We can even use the arthematic ops to go to the next date or previous date

Using the * :

Using * we can make the date go up to the multiples of today's date.

Example :

insert into timeExample values (Current_time(), current_date() * 3, NOW());

select * from timeExample;

Output :

times	dates	Today
23:26:44	2024-03-10	2024-03-10 23:26:44
23:28:41	6072-09-30	2024-03-10 23:28:41

Using the + ops :

We can make the date go up to the addition of the today's date upto the required number.

Example :

insert into timeExample values (Current_time(), current_date() + 3, NOW());

select * from timeExample;

Output :

times	dates	Today
23:26:44	2024-03-10	2024-03-10 23:26:44
23:28:41	6072-09-30	2024-03-10 23:28:41
23:31:05	2024-03-13	2024-03-10 23:31:05

Using the – ops :

We can make the date goto the previous date

Example :

insert into timeExample values (Current_time(), current_date() - 3, NOW());

select * from timeExample;

Output :

times	dates	Today
23:26:44	2024-03-10	2024-03-10 23:26:44
23:28:41	6072-09-30	2024-03-10 23:28:41
23:31:05	2024-03-13	2024-03-10 23:31:05
23:31:45	2024-03-07	2024-03-10 23:31:45

Functions / Group By clause;

In sql we are having some built in functions which will return the values by passing the parameters

1. Count :

Will return the no of data entries we are having in a column then we can use the count function.

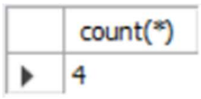
Syntax to get all the rows in table :

Select count(*) from tableName;

Example ;

select count(*) from customer;

Output :



	count(*)
▶	4

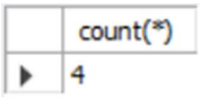
Syntax to get the count for a specific col:

Select count(colName) from tableName;

Example :

select count(id) as count from customer;

Output :



	count(*)
▶	4

2. Max :

Used to return the maximum value in a column.

Syntax ;

Select MAX(colName) from tableName;

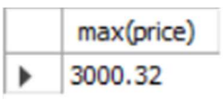
//for number will return the highest number

// for alphabet we use the last alphabet

Example :

select max(price) from products;

Output :



	max(price)
▶	3000.32

3. Min :

Used to return the minimum values in the column

Syntax :

Select min(colName) from tableName;

Example :

select min(price) from products;

Output :

	min(price)
▶	200.00

4. Avg :

Used to return the avg value for the column.

Syntax :

Select AVG(colName) from tableName;

Example :

SELECT AVG(price) as Average from products;

Output :

Average
1125.215000

5. Sum :

Used to return the sum value from the column

	id	name	sales
▶	1	mark	200
	2	mike	300
	3	sofie	400

Syntax :

Select SUM() from products;

Example :

SELECT sum(products.sales) from products;

Output :

	sum(products.sales)
▶	900

6. Concat :

Used to return the concat of two col values.

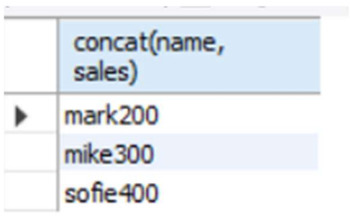
Syntax :

Select concat(col1, col2,..) from tableName;

Example :

SELECT concat(name, sales) from products;

Output :



	concat(name, sales)
▶	mark200
	mike300
	sofie400

Normalization :

The process of arranging data into database is called as normalization.

Here we collect the raw data and will arrange them into a table form by decomposing the values.

The main advantage of using the normalization is to remove the duplicate data (data redundancy) and to main data integrity.

Data Integrity refers to a consistent and accurate info through out the process. It makes sure that the data is not changed throughout the process.

The problems for the database :

1. Data anomalies :

a. Insertion anomalies :

Insertion of repeated data into a table is called as insertion anomalies.

b. Updated anomalies :

While updating the data if we miss matches any one the values incorrectly will lead to data inconsistency/ accuracy.

c. Deletion anomalies :

Here In a table we will have two or more tables depended on one another. Hence when we delete a table then the another table will also get deleted.

To eliminate the anomalies we use the normal forms.There are 5 normal forms

- ***1 NF***

- ***2 NF***
- ***3 NF***
- ***3.4 NF/ 4 NF***
- ***5 NF***

Lets look at the table which we are performing for the normalization.

emp_id	emp_name	emp_email	manager	dept	skills
1	Rock	rock@email.com	Shark	ocean	constant
2	Plant	plant@email.com	Sun	Sky	float
3	air	air@email.com	dust	earth	fly, wind
4	fire	fire@email.com	ash	forest	burn
5	water	water@email.com	sea	water	tsunami
6	Human	human@email.com	God	universe	walk, run, drink, eat, sleep

1st NF :

In first normal form we maintain the atomicity. This will help us to maintain the each column with individual value only. The main points it follows

a. It removes the repeating groups from table
b. Create a separate table for each set of data
c. Identify the each set of data with the primary key

Table after 1NF

emp_id	emp_name	emp_email	manager	dept	skills	Timing
1	Rock	rock@email.com	Shark	ocean	constant	am
2	Plant	plant@email.com	Sun	sky	float	pm
3	air	air@email.com	dust	earth	fly	am
4	fire	fire@email.com	ash	forest	burn	am
5	water	water@email.com	sea	earth	tsunami	graveyard
6	Human	human@email.com	God	earth	walk	am
3	air	air@email.com	dust	earth	wind	graveyard
6	Human	human@email.com	God	earth	run	pm
6	Human	human@email.com	God	earth	drink	pm
6	Human	human@email.com	God	earth	sleep	am
6	Human	human@email.com	God	earth	eat	graveyard

NOTE : 1st NF will only concentrate on only one thing that is it will not allow a row having multiple values. By default we use the 1st NF.

2nd NF :

It was defined by EF Codd. It follows the below conditions :

a. The table must satisfy the 1st NF.

b. There shouldn't be any partial dependency. If there then we need to make it fully dependent.

Partial Dependency :

Partial Dependency is having a functional dependency of a non-prime attribute to a prime attribute.

In our table the branch is defined by the as it is having relation with the manager and not with the emp. Hence we are separating the manager and dept from the table to make it fully dependent.

Table after performing 2NF :

mng_id	manager	dept
1	shark	ocean
2	sun	sky
3	dust	earth
4	ash	forest
5	sea	water
6	god	universe

emp_id	emp_name	emp_email	manager	skills	timings
1	Rock	rock@email.com	1	constant	am
2	Plant	plant@email.com	2	float	pm
3	air	air@email.com	3	fly	am
4	fire	fire@email.com	4	burn	am
5	water	water@email.com	5	tsunami	graveyard
6	Human	human@email.com	6	walk	am
3	air	air@email.com	3	wind	graveyard
6	Human	human@email.com	6	run	pm
6	Human	human@email.com	6	drink	pm
6	Human	human@email.com	6	sleep	am
6	Human	human@email.com	6	eat	graveyard

3rd NF :

It is used to remove the duplication of data and ensures the referential data integrity. It follows the below condition :

a. It must satisfy the 2nd NF.
b. There shouldn't be any transitive dependency.

Transitive dependency :

Here the non-primary key columns can't be functionally depends on the another non-primary key column. This will again lead to anomalies as it consisting the duplicate or repeated data.

In order to prevent it we split the table where we create a non-prime attributes to a separate table and prime attribute into a separate this way the non-prime attributes are not dependent to each other.

In above table we are having time where change in manager will change in time which will be an issue hence we separate the timing into separate table.

Table after performing 3NF :

timing_id	timings
t1	am
t2	pm
t3	graveyard

emp_id	emp_name	emp_email	manager	skills	timings
1	Rock	rock@email.com	1	constant	t1
2	Plant	plant@email.com	2	float	t2
3	air	air@email.com	3	fly	t1
4	fire	fire@email.com	4	burn	t1
5	water	water@email.com	5	tsunami	t3
6	Human	human@email.com	6	walk	t1
3	air	air@email.com	3	wind	t3
6	Human	human@email.com	6	run	t2
6	Human	human@email.com	6	drink	t2
6	Human	human@email.com	6	sleep	t1
6	Human	human@email.com	6	eat	t3

3.5NF :

It is called as Boyce Codd NF. It follows the below conditions.

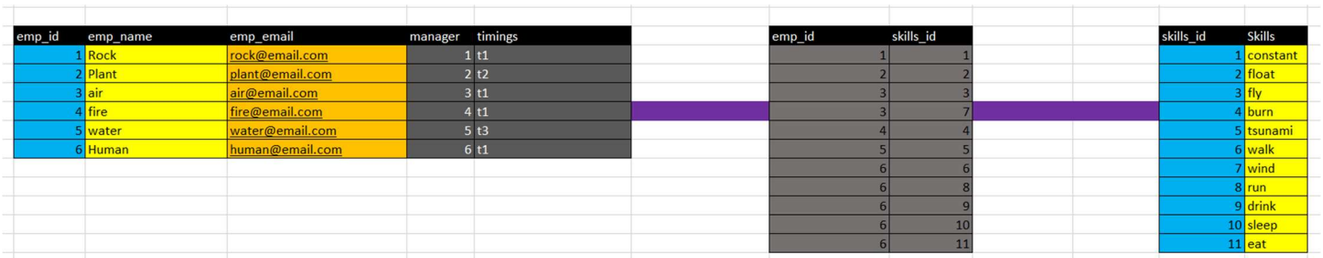
a. It must satisfy the 3rd NF
b. If there are 2 columns A and B. A is functionally dependent to B then the A must be a super key.

SuperKey :

Super key is used to identify the a row/ tuple in a table. A primary key can be a super key, but not all the super keys are primary key.

4th NF :

It is used to eliminate the multi value dependency between the columns. To do that we will split the table into multiple tables and connect them with another table in between them which will acts as a bridge.



emp_id	emp_name	emp_email	manager	timings
1	Rock	rock@email.com	1	t1
2	Plant	plant@email.com	2	t2
3	air	air@email.com	3	t1
4	fire	fire@email.com	4	t1
5	water	water@email.com	5	t3
6	Human	human@email.com	6	t1

emp_id	skills_id
1	1
2	2
3	3
3	7
4	4
5	5
6	6
6	8
6	9
6	10
6	11

skills_id	Skills
1	constant
2	float
3	fly
4	burn
5	tsunami
6	walk
7	wind
8	run
9	drink
10	sleep
11	eat

In the above table we having issues with the values that are repeating multiple times leading to the duplicates in database. Hence we separated

the column as separate one. And used another table to act as bridge between them.

Table before applying normalization :

emp_id	emp_name	emp_email	manager	dept	skills
1	Rock	rock@email.com	Shark	ocean	constant
2	Plant	plant@email.com	Sun	Sky	float
3	air	air@email.com	dust	earth	fly, wind
4	fire	fire@email.com	ash	forest	burn
5	water	water@email.com	sea	water	tsunami
6	Human	human@email.com	God	universe	walk, run, drink, eat, sleep

Table after applying Normalization :

emp_id	emp_name	emp_email	manager	timings
1	Rock	rock@email.com	1	t1
2	Plant	plant@email.com	2	t2
3	air	air@email.com	3	t1
4	fire	fire@email.com	4	t1
5	water	water@email.com	5	t3
6	Human	human@email.com	6	t1

Emp Table

skills_id	Skills
1	constant
2	float
3	fly
4	burn
5	tsunami
6	walk
7	wind
8	run
9	drink
10	sleep
11	eat

Skills Table

mng_id	manager	dept
1	shark	ocean
2	sun	sky
3	dust	earth
4	ash	forest
5	sea	water
6	god	universe

Manager Table

S	timings
t1	am
t2	pm
t3	graveyard

Shift Timings

Joins :

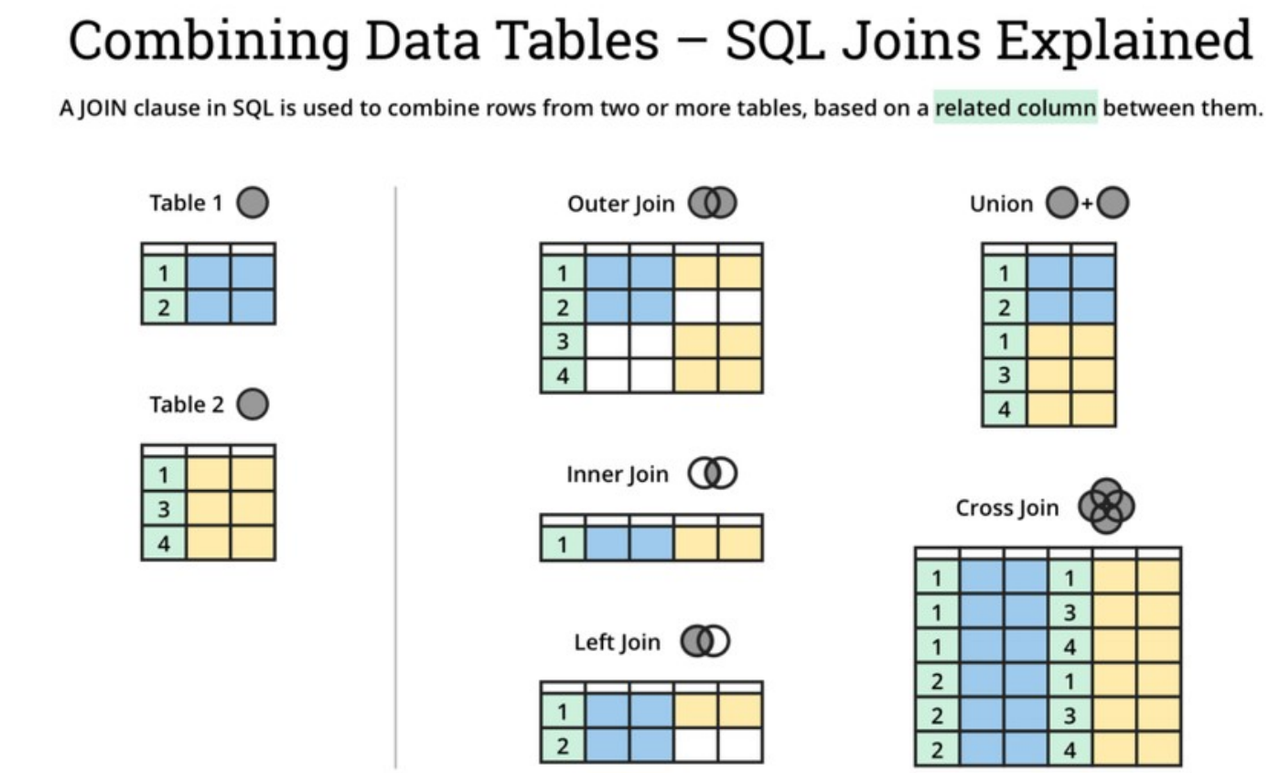
Joins are mainly used to combine different tables as a single table. Joins allow you to retrieve data from multiple tables simultaneously, enabling you to create complex queries that gather information from various sources.

Example Tables :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager
1	bod	9876543	bod@email.com	20900.000	M	1	1
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4
5	Asha	987654	asha@gmail.com	30000.000	F	2	5
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10
11	Kong	34567898	kong@email.com	40000.000	M	7	7

manager_id	manager_name	dept	location
1	patrick	software	Hyd
2	sai	hardware	remote
3	Tim	testing	Hybrid
4	Devika	developer	Hyd
5	Abhi	hr	Hyd
6	Sung	actor	Korea
7	Godzilla	destroyer	Amazon
8	Max	developer	Pune
9	Mahesh	developer	Chennai
10	Veve	Actor	South Korea
11	Mahesh	developer	Hyd

shift_id	timings
1	10am - 9pm
2	6pm - 3am
3	10am - 7pm
4	9am - 6pm
5	8am - 6pm
6	10am - 5pm
7	5pm - 6am
8	10pm - 9am



Example Tables :

Inner Join :

Here we join the two tables with the common data among them with the help of INNER JOIN keyword.

Syntax :

Select col/ uni selector from table(leftSide) INNER JOIN table(rightSide) ON condition

Example :

select * from emp as e INNER JOIN managerTable as mt ON e.manager = mt.manager_id;

Output :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	manager_id	manager_name	dept	location
1	bod	9876543	bod@email.com	20900.000	M	1	1	1	patrick	software	Hyd
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	sai	hardware	remote
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Tim	testing	Hybrid
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Devika	developer	Hyd
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Abhi	hr	Hyd
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	8	Max	developer	Pune
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6	6	Sung	actor	Korea
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	9	Mahesh	developer	Chennai
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	11	Mahesh	developer	Hyd
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Veve	Actor	South Korea
11	Kong	34567898	kong@email.com	40000.000	M	7	7	7	Godzilla	destroyer	Amazon

1. Left Join :

Will display everything on the left side of the table even though the data is not present on the right side. And also it will only follow the left table order and not the right table.

Syntax :

Select col/uniqueValue(univ) from table(leftSide) LEFT JOIN table(rightSide) ON condition;

Example :

select * from emp as e left JOIN managerTable as mt ON e.manager = mt.manager_id;

Output :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	manager_id	manager_name	dept	location
1	bod	9876543	bod@email.com	20900.000	M	1	1	1	patrick	software	Hyd
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	sai	hardware	remote
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Tim	testing	Hybrid
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Devika	developer	Hyd
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Abhi	hr	Hyd
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	8	Max	developer	Pune
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6	6	Sung	actor	Korea
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	9	Mahesh	developer	Chennai
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	11	Mahesh	developer	Hyd
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Veve	Actor	South Korea
11	Kong	34567898	kong@email.com	40000.000	M	7	7	7	Godzilla	destroyer	Amazon

2. Right Join :

It is complete opposite for the left join as it will display the entire data of the right side table even though the data didn't match. Also it will take the entire right table as major and will follow it's order.

Syntax :

Select col/ uni selector from table(leftSide) RIGHT JOIN table(rightSide) ON condition;

Example :

select * from emp as e right JOIN managerTable as mt ON e.manager = mt.manager_id;

Output :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	manager_id	manager_name	dept	location
1	bod	9876543	bod@email.com	20900.000	M	1	1	1	patrick	software	Hyd
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	sai	hardware	remote
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Tim	testing	Hybrid
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Devika	developer	Hyd
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Abhi	hr	Hyd
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6	6	Sung	actor	Korea
11	Kong	34567898	kong@email.com	40000.000	M	7	7	7	Godzilla	destroyer	Amazon
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	8	Max	developer	Pune
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	9	Mahesh	developer	Chennai
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Veve	Actor	South Korea
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	11	Mahesh	developer	Hyd

Self Join :

Self join is used to join the same table to itself.

emp_id	emp_name	emp_salary	emp_manager	branch	referal
1	Raju	10000.00	2	SEC	3
2	Rani	20000.00	1	HYD	5
3	Mani	40000.00	3	WAR	2
4	Venu	5000.00	1	HYD	1
5	Shena	500.00	2	SEC	4
6	Vicky	50000.00	3	WAR	3

Syntax :

Select * from table1 innerjoin table1 where condition;

Example :

select * from emp as e INNER JOIN emp as mt ON e.manager = mt.manager;

Output :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager
1	bod	9876543	bod@email.com	20900.000	M	1	1	1	bod	9876543	bod@email.com	20900.000	M	1	1
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	Guru	3495394	guru@gmail.com	30000.000	M	2	2
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Raju	593475394	raju@gmail.com	25000.000	M	8	3
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Asha	987654	asha@gmail.com	30000.000	F	2	5
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6	7	Yana	8765489	yana@gmail.com	37000.000	F	4	6
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10
11	Kong	34567898	kong@email.com	40000.000	M	7	7	11	Kong	34567898	kong@email.com	40000.000	M	7	7

NOTE : When using the self join we will face the ambiguity which leads to an error to rectify that we use the Alias names for the tables.

Syntax :

Table as aliasName;

We can select the only specific columns in self joins.

Syntax :

Select alias1.col1, alias1.col2 from table1 as alias innerjoin table1 as alias2 where condition;

Example :

select e.emp_name, mt.emp_email from emp as e INNER JOIN emp as mt ON e.manager = mt.manager;

Output :

emp_name	emp_email
bod	bod@email.com
Guru	guru@gmail.com
Raju	raju@gmail.com
Haritha	haritha@gmail.com
Asha	asha@gmail.com
Mini	mini19@gmail.com
Yana	yana@gmail.com
Yamuna	yamuna@gmail.com
Anusha	@gmail.com
Fanglin	fanglin@gmail.com
Kong	kong@email.com

Full Outer join:

In full outer join it will combine the two tables even thought the values didn't match. To do that we use the full join.

Syntax :

Select * from tableName full outer join table2 on condition;

NOTE : Full Outer Join is not possible in mysql.

Cross Join :

Cross join will display the final table as a product of total no of each rows. Example it table1 is having rows of 2 and table 2 is having 4 then it will return a total of 4 x 2 = 8 rows in total.

Syntax :

Select * from table as alias cross join table2;

Joining multiple tables :

Will join multiple tables as a single table.

Syntax :

Select */col from table1 join type table2 join type table3;

Example :

select * from emp as e Inner JOIN managerTable as mt inner JOIN shiftTable as st on e.shift_time = st.shift_id and e.manager = mt.manager_id order by e.emp_id ;

Output :

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	manager_id	manager_name	dept	location	shift_id	timings
1	bod	9876543	bod@email.com	20900.000	M	1	1	1	patrick	software	Hyd	1	10am - 9pm
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	sai	hardware	remote	2	6pm - 3am
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Tim	testing	Hybrid	8	10pm - 9am
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Devika	developer	Hyd	3	10am - 7pm
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Abhi	hr	Hyd	2	6pm - 3am
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	8	Max	developer	Pune	4	9am - 6pm
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6	6	Sung	actor	Korea	4	9am - 6pm
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	9	Mahesh	developer	Chennai	4	9am - 6pm
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	11	Mahesh	developer	Hyd	5	8am - 6pm
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Veve	Actor	South Korea	6	10am - 5pm
11	Kong	34567898	kong@email.com	40000.000	M	7	7	7	Godzilla	destroyer	Amazon	7	5pm - 6am

Group By :

We can use the group by to group a certain information.

emp_id	emp_name	salary	manager
1	June	4000.34	2
2	Dune	5000.32	4
3	Aune	5000.23	5
4	Bune	7000.12	1
5	Eune	9000.12	2
6	Lune	5000.32	4
7	Tune	5000.23	2
8	Pune	7000.12	1
9	Soon	9000.12	3
10	Noon	5000.32	2
11	Moon	5000.23	5
12	Loon	7000.12	1
13	Toon	9000.12	3

Syntax :

Select aggregate value/ selector from tableName where condition group by colName;

Example :

select sum(salary), manager from employee where manager > 2 group by manager;

Output :

sum(salary)	manager
18000.24	3
10000.64	4
10000.46	5

When ever you are using the group by you can use the having instead of where but to do that 1st we need to implement the group by then only we can use the having.

Syntax :

Select aggregate/ selector from tableName group by colName having condition;

Example :

select sum(salary), manager from employee group by manager having manager > 2;

NOTE : Most often we use the group by for the aggregate functions

Roll Up :

Roll up is an extension of the group by when we use the roll up it will create a new row by the end of the table. It is also known as super aggregate value.

Syntax :

Select selectors/ aggregate from tableName group by colName with rollup;

Example :

select sum(salary)from employee GROUP BY salary with rollup;

Output :

sum(salary)
4000.34
15000.69
15000.96
21000.36
27000.36
82002.71

Views :

Views are the virtual tables that are not actual columns inside the data base but a dummy table

To create views :

Syntax :

Create view viewName as select colName1, colName2 from table1;

Example :

create view sampleView as select emp_name, emp_id from employee;

Output :

To display view :

Syntax :

Select col/selector from viewTable;

Example :

select * from sampleView;

Output :

emp_name	emp_id
Raju	1
Rani	2
Mani	3
Venu	4
Shena	5
Vicky	6

To delete :

Syntax :

Drop view viewName;

Example :

Drop view sampleView;

Update view :

Syntax :

Create or replace view viewName as selectStatement;

Example :

create or replace view test as select emp_name, emp_id from employee;

NOTE: Views are auto updated. Hence if you make any changes in the table then the view is also get updated.

Indexes :

Index is a type of data structure. Which follows the B-Tree which is used to search the value quickly. As SQL uses the sequential method.

In index updating will take a lot of time as it needs to check all the conditions and req, though the selection will take less time.

emp_id	emp_name	emp_salary	emp_manager	branch	referal
1	Raju	10000.00	2	SEC	3
2	Rani	20000.00	1	HYD	5
3	Mani	40000.00	3	WAR	2
4	Venu	5000.00	1	HYD	1
5	Shena	500.00	2	SEC	4

To see the indexs :

Syntax :

Show index tableName;

Example :

show index from employee;

Output :

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
employee	0	PRIMARY	1	emp_id	A	6	NULL	NULL		BTREE			YES	NULL
employee	1	emp_manager	1	emp_manager	A	3	NULL	NULL	YES	BTREE			YES	NULL

To create index :

Syntax :

Create index indexName on tableName(colName);

Example :

create index test on employee(emp_id);

Types of Indexes :

1. *Single-column-index*
2. *Composite index*
3. *Unique index*
4. *Primary key index*
5. *Foreign key index*

In the context of databases, particularly relational database management systems (RDBMS) like MySQL, there are several types of indexes that serve different purposes in optimizing query performance and ensuring data integrity. Here are the primary types of indexes:

1. **Single-Column Index**:

An index created on a single column of a table. It speeds up searches and sorting operations based on that column.

Example :

create index sample_index on managers(manager_id);

Output :

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
managers	0	PRIMARY	1	manager_id	A	3	NULL	NULL		BTREE			YES	NULL
managers	1	sample_index	1	manager_id	A	3	NULL	NULL		BTREE			YES	NULL

2. **Composite Index**:

An index created on multiple columns of a table. It speeds up queries that involve filtering, sorting, or joining based on those columns.

Example :

create index compositeIndex on managers(manager_id, manager_name);

Output :

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
managers	0	PRIMARY	1	manager_id	A	3	NULL	NULL		BTREE			YES	NULL
managers	1	sample_index	1	manager_id	A	3	NULL	NULL		BTREE			YES	NULL
managers	1	compositeIndex	1	manager_id	A	3	NULL	NULL		BTREE			YES	NULL
managers	1	compositeIndex	2	manager_name	A	3	NULL	NULL	YES	BTREE			YES	NULL

3. **Unique Index**:

An index that ensures the uniqueness of values in one or more columns. It is similar to a single-column index but enforces uniqueness constraints.

4. **Primary Key Index**:

In many database systems, including MySQL, the primary key constraint is implemented using an index. It ensures the uniqueness of values in the primary key column(s) and serves as a fast lookup mechanism for retrieving rows by their primary key values.

5. **Foreign Key Index**:

In MySQL, creating a foreign key constraint also creates an index on the foreign key column(s) in the referencing table automatically.

Sub Queries :

We write a query inside another query then it is called as sub queries. The major advantages of writing the sub queries are we are reducing the hard coding and making it more dynamic.

Hard Coding :

Using a static value to the condition where the table and the value may change in future then we have to apply the changed/ updated value on the condition. If we are using the static value then it is called as Hard Coding.

Here we use the two types of queries

1. *Inner query*
2. *Outer query*

Always the inner query output will be the input for the outer query in sub queries.

manager_id	manager_name	dept	location
1	patrick	software	Hyd
2	sai	hardware	remote
3	Tim	testing	Hybrid
4	Devika	developer	Hyd
5	Abhi	hr	Hyd
6	Sung	actor	Korea
7	Godzilla	destroyer	Amazon
8	Max	developer	Pune
9	Mahesh	developer	Chennai
10	Veve	Actor	South Korea
11	Mahesh	developer	Hyd
12	Laddu	Sweet	India

shift_id	timings
1	10am - 9pm
2	6pm - 3am
3	10am - 7pm
4	9am - 6pm
5	8am - 6pm
6	10am - 5pm
7	5pm - 6am
8	10pm - 9am

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager
1	bod	9876543	bod@email.com	20900.000	M	1	1
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4
5	Asha	987654	asha@gmail.com	30000.000	F	2	5
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8
7	Yana	8765489	yana@gmail.com	37000.000	F	4	6
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10
11	Kong	34567898	kong@email.com	40000.000	M	7	7

Example for using sub query as a sub col:

select emp_id, emp_name, (select location from managerTable where managerTable.manager_id = emp.manager) as location from emp order by emp_id;

Output :

emp_id	emp_name	location
1	bod	Hyd
2	Guru	remote
3	Raju	Hybrid
4	Haritha	Hyd
5	Asha	Hyd
6	Mini	Pune
7	Yana	Korea
8	Yamuna	Chennai
9	Anusha	Hyd
10	Fanglin	South Korea
11	Kong	Amazon

Example for using the sub query as a condition :

select emp_id, emp_name, salary, (select AVG(salary) from emp) as avg_salary from emp where salary > (select AVG(salary) from emp);

Output :

emp_id	emp_name	salary	avg_salary
4	Haritha	40000.000	33445.4545455
6	Mini	35000.000	33445.4545455
7	Yana	37000.000	33445.4545455
10	Fanglin	50000.000	33445.4545455
11	Kong	40000.000	33445.4545455

There are 3 types of sub queries :

1. Scalar Subquery :

a. Will return one row and one column in general the above example is a scalar subquery as it is only returning one column and one row that is avg_salary.

Using scalar in joins :

select e1.emp_id, e1.emp_name, e1.salary, avg.sal from emp as e1 join (select AVG(e2.salary) as sal from emp as e2) as avg on e1.salary > avg.sal;

Output :

emp_id	emp_name	salary	sal
4	Haritha	40000.000	33445.4545455
6	Mini	35000.000	33445.4545455
7	Yana	37000.000	33445.4545455
10	Fanglin	50000.000	33445.4545455
11	Kong	40000.000	33445.4545455

2. Multiple Row Subquery :

a. It will return multiple values

b. It is having two types :

i. Multiple Columns and multiple rows
ii. Multiple Rows and one column

c. We need to use the IN keyword to check

Syntax for Multiple Columns and Multiple Rows:

Select col/ * from table where col IN (subquery);

We are using a combine column to show the example where we created views table for this with name all_vt.

emp_id	emp_name	emp_contact	emp_email	salary	gender	shift_time	manager	manager_id	manager_name	dept	location
1	Bud	9876543	bod@email.com	20900.000	M	1	1	1	patrick	software	Hyd
2	Guru	3495394	guru@gmail.com	30000.000	M	2	2	2	sai	hardware	remote
3	Raju	593475394	raju@gmail.com	25000.000	M	8	3	3	Tim	testing	Hybrid
4	Haritha	9284542	haritha@gmail.com	40000.000	F	3	4	4	Devika	developer	Hyd
5	Asha	987654	asha@gmail.com	30000.000	F	2	5	5	Abhi	hr	Hyd
6	Mini	12345674	mini19@gmail.com	35000.000	F	4	8	8	Max	developer	Pune
7	Yana	8765489	yana@gmail.com	37000.000	F	4	4	4	Devika	developer	Hyd
8	Yamuna	456789	yamuna@gmail.com	30000.000	F	4	9	9	Mahesh	developer	Chennai
9	Anusha	567890	anusha@gmail.com	30000.000	F	5	11	11	Mahesh	developer	Hyd
10	Fanglin	3456789	fanglin@gmail.com	50000.000	M	6	10	10	Veve	Actor	South Korea
11	Kong	34567898	kong@email.com	40000.000	M	7	7	7	Godzilla	destroyer	Amazon

Example :

select emp_id, emp_name, salary, dept from all_vt where (dept, salary) in (select dept, max(salary) from all_vt group by dept);

Output :

emp_id	emp_name	salary	dept
1	Bud	20900.000	software
2	Guru	30000.000	hardware
3	Raju	25000.000	testing
4	Haritha	40000.000	developer
5	Asha	30000.000	hr
10	Fanglin	50000.000	Actor
11	Kong	40000.000	destroyer

Example for Multiple rows and one column :

select manager_name from managertable where manager_id not in (select manager from emp group by manager);

Output ;

manager_name
Sung

Corelated Subquery :

The sub query is related to the outer query. For every outer query the inner query will run number of times.

Example :

select * from viewname as v1 where salary > (select avg(salary) from viewname as v2 where v2.dept = v1.dept);

Output :

emp_id	emp_name	salary	emp_contact	emp_email	gender	manager_name	dept	location	timings
4	Haritha	40000.000	9284542	haritha@gmail.com	F	Devika	developer	Hyd	10am - 7pm
6	Mini	35000.000	12345674	mini19@gmail.com	F	Max	developer	Pune	9am - 6pm
7	Yana	37000.000	8765489	yana@gmail.com	F	Devika	developer	Hyd	9am - 6pm

In the above example as you can in the inner query we are using v2 and v1 as reference as we are using the reference of the outer query we can't use it with out the outer query hence it is depended on the outer query.

NOTE : We don't use the correlated sub queries as it will execute a single query n no.of times for each outer query execution.

Case statement :

We can use a case statement when we are checking a certain condition.

Syntax :

```
Select col/ *, (
        Case when condition
        Then
            'statement'
        Else
            'statement'
        end
) from tableName;
```

Example :

```
select(
    case when salary > (select avg(salary) from viewname)
  then
        concat(emp_name, " having higher salary then avg salary")
    else
        concat(emp_name, " having less salary then avg salary")
```

```
end)
as caseColumn
from viewname;
```

Output :

caseColumn
Bud having less salary then avg salary
Guru having less salary then avg salary
Asha having less salary then avg salary
Haritha having higher salary then avg salary
Mini having less salary then avg salary
Yana having less salary then avg salary
Yamuna having less salary then avg salary
Anusha having less salary then avg salary
Fanglin having higher salary then avg salary
Sponge having higher salary then avg salary
Kong having higher salary then avg salary
Raju having less salary then avg salary

Using Having clause :

We can use the having clause in the sub queries.

Example :

```
select dept, sum(salary) as total_salary,
(select avg(salary) from viewname) as average_salary from
viewname group by dept
having
sum(salary) > (select avg(salary) from viewname);
```

Output :

dept	total_salary	average_salary
developer	172000.000	37608.7500000
Actor	50000.000	37608.7500000
Sweet	83405.000	37608.7500000
destroyer	40000.000	37608.7500000

Using the subqueries to insert information :

We can use the subqueries while inserting the information into the table. Let talk about an example where are trying to insert emp information into another table and we don't want to insert duplicate table hence we use the subqueries.

Creating emp_history table ;

```
create table emp_history(
    emp_id int,
```

```
    emp_name varchar(40),
    dept varchar(30),
    manager_name varchar(40),
    salary decimal,
    location varchar(30)
);
```

Example :

```
insert into emp_history
select
    e1.emp_id,
    e1.emp_name,
    m1.dept,
    m1.manager_name,
    e1.salary,
    m1.location
from emp as e1
join
managertable as m1
on e1.manager = m1.manager_id
where
not exists (
    select 1 from emp_history eh where e1.emp_id =
    eh.emp_id
);
```

Output :

emp_id	emp_name	dept	manager_name	salary	location
1	Bud	software	patrick	20900	Hyd
2	Guru	hardware	sai	30000	remote
3	Raju	testing	Tim	25000	Hybrid
4	Haritha	developer	Devika	40000	Hyd
5	Asha	hr	Abhi	30000	Hyd
6	Mini	developer	Max	35000	Pune
7	Yana	developer	Devika	37000	Hyd
8	Yamuna	developer	Mahesh	30000	Chennai
9	Anusha	developer	Mahesh	30000	Hyd
10	Fanglin	Actor	Veve	50000	South Korea
11	Kong	destroyer	Godzilla	40000	Amazon
13	Sponge	Sweet	Laddu	83405	India

Using subqueries for updating :

Example :

```
update emp_history as e set salary =
        (select (max(salary) * 0.5) from emp where
    emp.emp_id = e.emp_id)
where
    e.dept in (select dept from managertable where location =
    "Hyd")
and
    e.emp_id in (select emp_id from emp_record);
```

Output :

emp_id	emp_name	dept	manager_name	salary	location
1	Bud	software	patrick	10450	Hyd
2	Guru	hardware	sai	30000	remote
3	Raju	testing	Tim	25000	Hybrid
4	Haritha	developer	Devika	20000	Hyd
5	Asha	hr	Abhi	15000	Hyd
6	Mini	developer	Max	17500	Pune
7	Yana	developer	Devika	18500	Hyd
8	Yamuna	developer	Mahesh	15000	Chennai
9	Anusha	developer	Mahesh	15000	Hyd
10	Fanglin	Actor	Veve	50000	South Korea
11	Kong	destroyer	Godzilla	40000	Amazon
13	Sponge	Sweet	Laddu	83405	India

Using subqueries for delete :

Example :

```
delete from managertable where manager_id not in (select manager
from emp) ;
```

Output :

manager_id	manager_name	dept	location
1	patrick	software	Hyd
2	sai	hardware	remote
3	Tim	testing	Hybrid
4	Devika	developer	Hyd
5	Abhi	hr	Hyd
7	Godzilla	destroyer	Amazon
8	Max	developer	Pune
9	Mahesh	developer	Chennai
10	Veve	Actor	South Korea
11	Mahesh	developer	Hyd
12	Laddu	Sweet	India

On delete :

Will define the child row what needs to do if the foreign key is deleted from the parent table.

parent_id	parent_id	child_id
1	1	11
2	2	12
3	3	13
4	4	14
5	5	15

Syntax :

Create table tableName (colName, foreign key (childColName) references parentTableName(parentColName) on delete option);

There are several options we can use when the parent table deleted.

1. **Cascade :**
   a. Will delete the entire row

   Syntax :

   Create table tableName (colName, foreign key (childColName) references parentTableName(parentColName) on delete cascade);

   Example :

```
create table child(
        parent_id int default 0,
    child_id int primary key,
    foreign key (parent_id) references parent(parent_id) on delete
    CASCADE
);
```

Deleting the value :

parent_id
1
3
4
5

parent_id	child_id
1	11
3	13
4	14
5	15
NULL	NULL

2. **Set null :**
   a. Will replace the value with the null

Syntax :

Create table tableName (colName, foreign key (childColName) references parentTableName(parentColName) on delete cascade);

Example :

```
create table child(
parent_id int default 0,
child_id int primary key,
foreign key (parent_id) references parent(parent_id) on delete set null
);
```

Output :

parent_id
1
3
4
5

parent_id	child_id
NULL	12
1	11
3	13
4	14
5	15

3. **Restrict :**
   a. Will restrict the parent table from deleting the foreign key
4. **No action :**
   a. similar to restrict, just used for clarity.

NOTE : By default it will use the restrict.

To drop a foreign key we need to use the alter command.

Syntax :

Alter table tableName drop foreign key foreignKey ;

To update a foreign key we use the alter and add constraint.

Syntax :

Alter table tableName add constraint foreignKeyName foreign key (colName) references parent(colName) on delete option;

Stored Procedures :

It is a process of storing the sql statements and can be re-used.

To create a procedure :

Syntax :

```
Create procedure statementName()
Begin
    Query
End;
```

NOTE: As the semi-colon is a delimiter we need to use it in two locations at the query and also at the end. Hence It will throw an error to rectify this we use the delimiter

Syntax :

```
Delimiter $$ or //
Create procedure procedureName()
BEGIN
    Query;
END $$
Delimiter ;
```

Example for creating stored procedure :

```
Delimiter $$
create procedure display()
begin
    select * from TEST;
end $$
Delimiter ;
```

To call a stored procedure :

To call a stored procedure we use the call keyword before the stored procedure name.

Syntax :

```
CALL procedureName();
```

Example :

```
call display();
```

Output :

D	T	DT
2024-03-21	18:11:51	2024-03-21 18:11:51
1000-12-11	12:00:00	1000-12-11 12:00:00

To delete the store procedure :

To delete we use the drop statement inorder to remove it.

Syntax :

```
Drop procedure procedureName;
```

Example :

```
drop PROCEDURE display;
```

To display using a condition :

To make the procedure use the condition then follow the below syntax

Syntax :

```
Delimiter $$
Create procedure procName( IN value DataType)
BEGIN
Query;
END $$
Delimiter ;
```

Example :

```
delimiter $$
create procedure findById( in D date)
begin
select * from TEST as T where T.D = D;
end$$
delimiter ;
```

Output :

D	T	DT
2024-03-21	18:11:51	2024-03-21 18:11:51

NOTE : If we requires multiple parameters then we use the IN ParameterName DataType.

NOTE : We can only pass the colNames and not the tableName or datatype.

NOTE : We use the new keyword to make the mysql work on the new values.

Hall of Fame :

1. Safe_updates :

a. If you facing issue when inserting the data using where condition then we use the sql_safe_updates to enable or disable.

Syntax :

Update table set sql_safe_update = 0; // used to disable safemode
Update table set sql_safe_update = 1; // used to enable safemode

2. Read only :

a. If you don't want to change the values into the table then we use the read only property.

Syntax :

Alter database dataBaseName read only = 1;// disable readonly mode

Alter database dataBaseName readonly = 0; // disable readonly mode

3. Set auto commit :

a. Generally we the sql will have the auto commit option by default, we can turn it on and off by using this query.

Syntax :

Set AUTOCOMMIT = OFF/ ON;

4. After :

It is used to add a table after a column while we are adding new column to an existing table.

Syntax :

Alter table tableName add column colName datatype after existingCol;

Example :

alter table tableName add column col3 varchar(30) after col1;

5. Distinct :

It is a keyword used to return a unique values from the table.

Syntax :

Select distinct colName from tableName;

emp_id	name	salary	area	tl
1	MSD	150.100	Delhi	3
2	VK	200.500	Mumbai	2
3	DK	100.400	Hyd	4
4	KL	900.000	Mah	5
5	UL	1000.000	AP	1
6	RL	1000.000	Assam	2
7	Yuvi	1000.000	Kolkata	5
8	Bhuvi	1000.000	Goa	1

Example :

Select distinct salary from emp;

Output :

salary
150.100
200.500
100.400
900.000
1000.000