Section 1: Introduction and Terminologies

Terminologies Used in Databases

In databases, data is stored in the form of tables. For example, consider two tables named **Student** and **Dept**. These tables are also referred to as **Relations**.

That's why a **DBMS (Database Management System)** is also known as an **RDBMS (Relational Database Management System)**—because data is stored in **relational** structures, i.e., tables.

Relation: As shown below, the tables **Student** and **Dept** are called *Relations* in the context of databases.

Table1			
Student			
Roll	Name	City	Deptno
1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20
4	Ramesh	Delhi	30
5	Suneeta	Lucknow	40
6	Anita	Kolkata	30
7	Raj	Jaipur	30
8	Ali	Lucknow	40
9	Michael	Cochin	10
10	Pavan	Vijayawada	20
11	Suraj	Hyderabad	10
12	Altaf	Bengaluru	40
13	Ravi	Indore	20
14	Verma	Delhi	20
15	Sharma	Vizag	10

Table2	
Dept	
Deptno	Name
10	CSE
20	ECE
30	Civil
40	Mech

Terminology
1. Relation
2.Record
3. Field
4. Primary Key
5. Relationship
6. Foreign Key
7. Constraints

Section 2: Database Setup Overview

Setting Up the Database

We will now create tables in the database that we previously set up and insert sample data into them.

Earlier, we saw how to install the **SQLite** database and how to create a new database—let's call it **university**.

- **In the `university` database, we will create tables and insert data.**

- **To perform operations like creating, inserting, and retrieving data, we must learn SQL.**

- **We will explore SQL concepts with practical examples using the university database.**

Section 3: SQL Command Types (DDL, DML, Query Language)

Types of SQL Commands

Let's start by learning the different types of SQL commands.

SQL is generally divided into the following categories:

1. **DDL (Data Definition Language) – Used to define and modify the structure of database objects like tables.**

2. **DML (Data Manipulation Language) – Used for inserting, updating, and deleting data.**

3. **Query Language – Used for retrieving data using the `SELECT` command.**

4. **DCL (Data Control Language) – Used for permissions and access control (e.g., `GRANT`, `REVOKE`).**

Note: You can explore more about DCL via official SQL documentation or online resources.

SQLite		
DDL	DML	Query
1. CREATE	1. INSERT	select
2. DROP	2. DELETE	from
3. ALTER	3. UPDATE	join
4. TRUNCATE		where
5. RENAME		order by
		group by
		having

Section 4: DDL – Data Definition Language

DDL – Data Definition Language

The following commands come under **DDL**:

1. **CREATE** – Used to create a new table.
2. **DROP** – Used to delete an existing table.
3. **ALTER** – Used to modify the structure of an existing table.
4. **TRUNCATE** – Used to delete all rows from a table, while retaining the structure.
5. **RENAME** – Used to rename a table (rarely used).

Most commonly, we use the `CREATE` command. Once the database schema is finalized and the necessary tables are defined, changes are seldom made.

This is because we typically ensure that the database structure is well-designed before creating the actual tables.

The remaining commands (i.e., **DROP, ALTER, TRUNCATE, RENAME**) are rarely used, but they are part of the **Data Definition Language (DDL).**

When we create a table, we are essentially defining its structure — that is, providing a description of the table. This includes specifying the columns, their names, data types, and other related properties.

Once the structure is defined, we can **insert** data into the table — meaning we add **rows**.

So, DDL is used to define the table structure (headers), whereas DML (Data Manipulation Language) is used to insert, update, or delete data within the table.

Section 5: DML – Data Manipulation Language

DML
1. INSERT
2. DELETE
3. UPDATE

DML – Data Manipulation Language

DML is used to modify the actual data in the tables.

Common DML commands include:

1. **INSERT** – Adds a new row to a table.
2. **DELETE** – Removes a row from a table.
3. **UPDATE** – Modifies the data in existing rows.

- **We use DDL to define the structure of tables (headers/columns).**

- **We use DML to insert, delete, or update records in those tables.**

Section 6: Query Language – SELECT

Query Language – SELECT

The **SELECT** command is part of the query language and is used to retrieve data.

When we create a table, we define what columns (headers) we need. For example, a `rollno` column would typically have an `INTEGER` datatype, while `name` would be a `TEXT` (String) type.

It is essential to understand **data types** supported by your database system.

We are using **SQLite**, which is loosely typed but still allows us to define and enforce datatypes for consistency.

Creating a Table

When we create a table, we define its structure — specifically the **headers** or **columns** it will contain. This means specifying which columns are required.

For example, if we need a `rollno` and a `name`, we must also define the **data type** for `rollno`. Is it an **integer**, a **float**, a **numeric** type (with or without decimals), or a **string**? We must clearly specify the type of each column.

So, while defining columns, we also need to define their **respective data types**. This makes it important to understand what data types are supported by the database we're working with.

In this section, we'll focus on the data types available in **SQLite**.
Keep in mind that data types can vary slightly between databases — for example, between **MySQL**, **SQL Server**, and **SQLite**. These differences are minor and easy to understand or adapt to.

Therefore, whenever you use any database, it's essential to be familiar with the **data types supported by that specific system**.

Now, let's explore the data types available in **SQLite**.
Note that SQLite is considered **typeless**, meaning it allows storing **any type of data** in a column. However, maintaining **uniformity** is important. For example, if one row has `rollno = 10` and another has `rollno = 'ab'`, it leads to inconsistency.

Thus, it becomes **important for a programmer or database designer to define appropriate data types**. While SQLite allows storing different types of values, explicitly defining column data types is a better practice.

Let's take a look at the data types supported in SQLite.

Section 7: SQLite Data Types

DATA Types in SQLLite:

Data Types of SQLite	Description
NULL	Values not Present. We don't leave empty. We write something so if value is not present we write NULL.
INTEGER	Like Java Integer. It will contain numeric value without decimal.
REAL/FLOAT/NUMERIC	How many digits before and after decimal we can mention that. ex: (5,2)
TEXT	Text is nothing but same as String in Java.
CHAR/VARCHAR	String can also be taken as CHAR/VARCHAR. So we can take TEXT/CHAR/VARCHAR all represent String.
BLOB	Binary Large Object. This datatype is useful for storing a row data.Suppose we have an image, or we have a Video or we have an Audio. Ex: If we want to store an image. (image have .png/.jpeg/.gif etc format). We should store. We have to store the content of this file as a row data. i.e. just Binary Data. Coversion process will be reuired while fethcing the data.

Data Types in MySQL:

Summary of Common Data Types:

Type	Description
`TINYINT`	Small integer
`INT`	Standard integer
`BIGINT`	Large integer
`DECIMAL`	Exact numeric values
`FLOAT`, `DOUBLE`	Floating-point numbers
`CHAR`, `VARCHAR`	Fixed or variable-length strings
`TEXT`, `BLOB`	Large text or binary data
`DATE`, `DATETIME`	Date and time values
`ENUM`, `SET`	Predefined set of values
`JSON`	JSON data
`UUID`	Universally Unique Identifier

If we are taking any other database then we should be familiar with the datatype available in that database, which is a simple task.

Section 8: MySQL Data Types and Table Creation Steps

So what all the task we have to do for creating table:

Create tables and Insert data

1. Create Database
2. Create Table
3. Define Keys
4. Insert Data
5. Query Data

Create Database: First thing is to cerate a database. We have already created a database previously. Now we have to create table.

We are going to use sglite3

1. We are going to use these in MACos
2. First we need to move to the location where we have kept sglite after extracting like below.
   navin@Navins-MacBook-Pro downloads % **cd/Applications/SQLite**
3. Then we will hit the command as below
   navin@Navins-MacBook-Pro SQLite % **sqlite3**

Which should give the information as below if sqlite is successfully installed.

SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
Connected to a **transient in-memory database**.
Use ".open FILENAME" to reopen on a persistent database.

4. **No Database File Specified**

   When you open the SQLite shell (`sqlite3`) without specifying a database file, SQLite starts with a **transient in-memory database**, which doesn't persist data and has no file associated with it.

   This is why `.databases` shows `main: ""`, meaning no physical file is currently attached.

   sqlite> **.databases**
   main: "" r/w

5. Now, Hit below **command to create a new database**.
   sqlite> **.open university.db**

6. Hit **.databases** command to see the database create. Once the database is created successfully, which will give us details of database created as below.
   sqlite> **.databases**
   main: /Applications/SQLite/university.db r/w

Now Create a table

To verify if any table is present in the university databse/any database

Note: If we will not have any table in database we will not get any table info

```
sqlite> .tables
```

Command1:

```
sqlite> .tables
```

So we should create a table.

Command2:

```
sqlite> create table dept(deptno integer primary key not null unique,dname text);
sqlite> .table
dept
```

We require two tables. So we have created table one as **dept** and another one is **students**.

Create Dept Table: **create table dept(deptno integer primary key not null unique,dname text);**

Insert data into table: We are inserting 5 records in **dept** table

```
insert into dept(deptno,dname) values(10,'CSE');
insert into dept(deptno,dname) values(20,'ECE');
insert into dept(deptno,dname) values(30,'CIVIL');
insert into dept(deptno,dname) values(40,'Mech');
insert into dept(deptno,dname) values(40,'TCE');
```

```
mysql> CREATE TABLE DEPT(DEPTNO INTEGER PRIMARY KEY NOT NULL UNIQUE, NAME TEXT);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(10, CSE);
ERROR 1054 (42S22): Unknown column 'CSE' in 'field list'
mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(10, 'CSE');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(20, 'ece');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(30, 'CIVIL');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(40, 'MECH');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO DEPT(DEPTNO,NAME) VALUES(50, 'TCE');
Query OK, 1 row affected (0.00 sec)

mysql> SHOW TABLES
    -> ;
+----------------+
| Tables_in_dept |
+----------------+
| dept           |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM DEPT;
+--------+-------+
| DEPTNO | NAME  |
+--------+-------+
|     10 | CSE   |
|     20 | ece   |
|     30 | CIVIL |
|     40 | MECH  |
|     50 | TCE   |
+--------+-------+
```

Integrity constraint: Types of Integrity Constraints

Here are the commonly used integrity constraints in SQL:

1. PRIMARY KEY Constraint

- **Ensures that a column (or set of columns) has unique and non-null values, uniquely identifying each record in the table.**

Ex: create table dept(deptno integer primary key not null unique,dname text);

2. FOREIGN KEY Constraint

- **Enforces a relationship between two tables by ensuring that the value in a column (or set of columns) matches a value in the referenced table's primary key.**

Ex: create table student(rollno integer primary key not null unique, name text, city text, deptno integer,foreign key(deptno) references(deptno));

Note: 1. deptno in student table we have considered as foreign key it means, deptno should be present in dept table and that should be the primary key.

2. While inserting records in student table we need to make sure that we can enter only those values as foreign key which are already present as primary key of dept table.

Ex: We have deptno 10,20,30,40 and 50 in dept table.

So, we can't insert a record for foreign key in student which are not available in deptno of dept table. Ex: 60,70,80 etc. It should be only 10,20,30,40,50.

3. UNIQUE Constraint

- **Ensures that all values in a column (or set of columns) are unique but allows a single `NULL` value (unlike `PRIMARY KEY`).**

Example:

```
CREATE TABLE users (
    user_id INT,
    email VARCHAR(100) UNIQUE
);
```

4. NOT NULL Constraint

- **Ensures that a column cannot have `NULL` values.**

Example:

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50) NOT NULL
```

-);

Create student table: **create table student(rollno integer primary key not null unique, name text, city text, deptno integer,foreign key(deptno) references(deptno));**

Note: foreign key won't be activated bydefault in SQL Lite. So, we need to do that forcefully. To do that hit the command as below.
command:
pragma foreighn_key=ON;

Insert record into student Table: We are inserting 15 records in **student** table.

```
insert into student(rollno,name,city,deptno) values(1,'Ajay','Delhi',10);
insert into student(rollno,name,city,deptno) values(2,'Vijay','Kolkata',10);
insert into student(rollno,name,city,deptno) values(3,'Ajay','Mumbai',20);
insert into student(rollno,name,city,deptno) values(4,'Ramesh','Delhi',30);
insert into student(rollno,name,city,deptno) values(5,'Suneeta','Lucknow',40);
insert into student(rollno,name,city,deptno) values(6,'Anita','Kolkata',30);
insert into student(rollno,name,city,deptno) values(7,'Raj','Jaipur',30);
insert into student(rollno,name,city,deptno) values(8,'Ali','Lucknow',40);
insert into student(rollno,name,city,deptno) values(9,'Michael','Cochin',10);
insert into student(rollno,name,city,deptno) values(10,'Pavan','Vijayawada',20);
insert into student(rollno,name,city,deptno) values(11,'Suraj','Hyderabad'10);
insert into student(rollno,name,city,deptno) values(12,'Altaf','Bengaluru',40);
insert into student(rollno,name,city,deptno) values(13,'Ravi','Indore',20);
insert into student(rollno,name,city,deptno) values(14,'Verma','Delhi',20);
insert into student(rollno,name,city,deptno) values(15,'Sharma','Vizag',10);
```

```
mysql> CREATE TABLE STUDENTS(rollno integer primary key, name text, city text, deptno integer);
Query OK, 0 rows affected (0.03 sec)

mysql> insert into students(rollno, name, city, deptno) values(1,Ajay, Delhi, 10);
ERROR 1054 (42S22): Unknown column 'Ajay' in 'field list'
mysql> insert into students(rollno, name, city, deptno) values(1, 'Ajay', 'Delhi', 10);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(2, 'Vijay', 'Kolkata', 10);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(3, 'Ajay', 'Mumbai', 20);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(4, 'Ramesh', 'Delhi', 30);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(4, 'Suneeta', 'Lucknow', 40);
ERROR 1062 (23000): Duplicate entry '4' for key 'students.PRIMARY'
mysql> insert into students(rollno, name, city, deptno) values(5, 'Suneeta', 'Lucknow', 40);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(6, 'Anita', 'Kolkata', 30);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(7, 'Raj', 'Jaipur', 30);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(8, 'Ali', 'Lucknow', 40);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(9, 'Michael', 'Cochin', 10);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(10, 'Pavan', 'Vijayawada', 20);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> use dept;
Database changed
mysql> insert into students(rollno, name, city, deptno) values(11, 'Suraj', 'Hyderabad', 10);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(12, 'Altaf', 'Bengaluru', 40);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students(rollno, name, city, deptno) values(12, 'Ravi', 'Indore', 20);
ERROR 1062 (23000): Duplicate entry '12' for key 'students.PRIMARY'
mysql> insert into students(rollno, name, city, deptno) values(13, 'Ravi', 'Indore', 20);
Query OK, 1 row affected (0.00 sec)

mysql> insert into students(rollno, name, city, deptno) values(14, 'Verma', 'Delhi', 20);
Query OK, 1 row affected (0.00 sec)

mysql> insert into students(rollno, name, city, deptno) values(15, 'Sharma', 'Vizag', 10);
Query OK, 1 row affected (0.01 sec)

mysql> show table;\
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''
 at line 1
mysql> show table;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''
 at line 1
mysql> show tables;
+----------------+
| Tables_in_dept |
+----------------+
| dept           |
| students       |
+----------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from students;
+--------+---------+------------+--------+
| rollno | name    | city       | deptno |
+--------+---------+------------+--------+
|      1 | Ajay    | Delhi      |     10 |
|      2 | Vijay   | Kolkata    |     10 |
|      3 | Ajay    | Mumbai     |     20 |
|      4 | Ramesh  | Delhi      |     30 |
|      5 | Suneeta | Lucknow    |     40 |
|      6 | Anita   | Kolkata    |     30 |
|      7 | Raj     | Jaipur     |     30 |
|      8 | Ali     | Lucknow    |     40 |
|      9 | Michael | Cochin     |     10 |
|     10 | Pavan   | Vijayawada |     20 |
|     11 | Suraj   | Hyderabad  |     10 |
|     12 | Altaf   | Bengaluru  |     40 |
|     13 | Ravi    | Indore     |     20 |
|     14 | Verma   | Delhi      |     20 |
|     15 | Sharma  | Vizag      |     10 |
+--------+---------+------------+--------+
15 rows in set (0.00 sec)

mysql>
```

As we can see above we have created 2 tables, dept and students.

DELETE Record from a table:

Ex: **select * from dept;**

10|CSE
20|ECE
30|Civil
40|Mech
50|IT
60|

As we can see we have deptno=60. We will delete this.

delete from dept where deptno='60';
select * from dept;
10|CSE
20|ECE
30|Civil
40|Mech
50|IT

UPDATE Record in a table:

Ex: update student SET city = 'Bengaluru' WHERE rollno = 16 and name='Navin' and deptno=50;

Before Update the value was: 16|Navin|Bangalore|50
After Update query the is value is: 16|Navin|Bengaluru|50

One more sample of update query from mg_users_flat of mygate. Where we are updating value from deleted=0 to deleted=1.

update mg_users_flat set deleted=1 where mobile = 9590727199 and society_id=440;

Table1			
Student			
Roll	Name	City	Deptno
1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20
4	Ramesh	Delhi	30
5	Suneeta	Lucknow	40
6	Anita	Kolkata	30
7	Raj	Jaipur	30
8	Ali	Lucknow	40
9	Michael	Cochin	10
10	Pavan	Vijayawada	20
11	Suraj	Hyderabad	10
12	Altaf	Bengaluru	40
13	Ravi	Indore	20
14	Verma	Delhi	20
15	Sharma	Vizag	10

Table2	
Dept	
Deptno	Name
10	CSE
20	ECE
30	Civil
40	Mech

Clauses
select
from
join
where
order by
group by
having

SQL: SQL query language/Data retrieval language/Select command or clause

Basically when we say query language, it means we are talking about Data Retrieval Language.

So, below are the clauses that are used for writing queries for retrieval of data.

Clauses
select
from
join
where
order by
group by
having

Queries is written using below clauses and they are written in the same order

1. select 2. from 3. join 4. where 5. order by 6. group by 7. having

We will be using above two tables, i.e. student and dept

select: Along with **select** we must use **from** also. Without that we can't form a complete query.

select is used for specifying the columns of a relation that we want to retrieve, means we have to mention the column names. You want roll or you want name, what you want you have to mention that.

from is used for specifying a relation name i.e. table name. You wanted from Student table or a Dept table. So we have to use these two together.

Queries for Select and from:

Ex1: select name from students;

It means we want to select column **name** from **students** table/students Relation. So, it will show all the names of the table. So we have 15 name we have already inserted so it will show all the names.

Ajay
Vijay
Ajay
Ramesh
Suneeta
Anita
Raj
Ali
Michael
Pavan
Suraj

Altaf
Ravi
Verma
Sharma

Ex2: select name,rollno from students;
As we can see below, its showing both Ajayi|10 and Ajay|20. The reason is the entire row is not duplicate. They are distinct in terms of deptno they are distinct. Suppose, Ajay|20 dept no is also 10 then it will show only one.
So when we are using multiple columns, so it will treat two rows as duplicate only if all the values are same. Not just one value. Not just name or not just deptno. If both are same then it's duplicate. So usually we use distinct only with one column. And we use multiple columns only if there is a possibility of duplicate in all the columns.

Ajay|10
Vijay|10
Ajay|20
Ramesh|30
Suneeta|40
Anita|30
Raj|30
Ali|40
Michael|10
Pavan|20
Suraj|10
Altaf|40
Ravi|20
Verma|20
Sharma|10

Ex3: When we want to show all the columns we don't have to mention all the columns we can specify ***.**
***** means we want all the columns from students columns. So it will show all the rows, all the complete data. So we will get the table.

```
select * from students;
```

1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20
4	Ramesh	Delhi	30
5	Suneeta	Lucknow	40
6	Anita	Kolkata	30
7	Raj	Jaipur	30
8	Ali	Lucknow	40
9	Michael	Cochin	10
10	Pavan	Vijayawada	20
11	Suraj	Hyderabad	10
12	Altaf	Bengaluru	40
13	Ravi	Indore	20
14	Verma	Delhi	20
15	Sharma	Vizag	10

Ex4: Now similarly we write

select * from dept;

So we will get all the rows of dept table. So it will show only two columns as there are only two columns in dept table.

10	CSE
20	ECE
30	Civil
40	Mech

Ex5: select deptno from students;

10
10

20
30
40
30
30
40
10
20
10
40
20
20
10

Ex6: select **distinct** deptno from students;
Note: Generally, we use distinct to fetch the data for one column.

10
20
30
40

I.e. Unique deptno. Lets say, 10 is repeating many times, so it will appear only one time.

Ex7: select distinct name from students;

Ajay will be shown only once. After Ajay In between Vijay and Ramesh one more Ajay was there. So it is removed. Because we set distinct. So, Along with select clause we can use distinct and we can get just the unique names or unique values.

Ajay

Vijay
Ramesh
Suneeta
Anita
Raj
Ali
Michael
Pavan
Suraj
Altaf
Ravi
Verma
Sharma

where: where clause is used for specifying condition upon any field. One or more field or one or more column. So, where clause is used for filtering rows. Means we can remove some rows and show only those rows which are satisfying the condition or predicate.

select clause is used for filtering column, where clause is used for filtering rows. And showing only those rows which meets the condition. Let's see example to use where

Operator	Query Example	Explanation
<	SELECT * FROM Students WHERE rollno < 5;	Retrieves all rows where rollno is less than 5.
<=	SELECT * FROM Students WHERE rollno <= 5;	Retrieves all rows where rollno is less than or equal to 5.
>	SELECT * FROM Students WHERE rollno > 10;	Retrieves all rows where rollno is greater than 10.
>=	SELECT * FROM Students WHERE rollno >= 10;	Retrieves all rows where rollno is greater than or equal to 10.
=	SELECT * FROM Dept WHERE Name = 'CSE';	Retrieves rows from Dept where Name is

		"CSE."
<> (or !=)	SELECT * FROM Students WHERE city <> 'Delhi';	Retrieves rows where city is not "Delhi."
AND	SELECT * FROM Students WHERE city = 'Delhi' AND deptno = 10;	Retrieves rows where city is "Delhi" and deptno is 10.
OR	SELECT * FROM Students WHERE city = 'Delhi' OR deptno = 20;	Retrieves rows where city is "Delhi" or deptno is 20.
NOT	SELECT * FROM Students WHERE NOT city = 'Delhi';	Retrieves rows where city is not "Delhi."
LIKE	SELECT * FROM Students WHERE name LIKE 'A%';	Retrieves rows where name starts with "A."
BETWEEN	SELECT * FROM Students WHERE rollno BETWEEN 5 AND 10;	Retrieves rows where rollno is between 5 and 10 (inclusive).
IN	SELECT * FROM Students WHERE deptno IN (10, 20);	Retrieves rows where deptno is either 10 or 20.
NOT IN	SELECT * FROM Students WHERE deptno NOT IN (10, 20);	Retrieves rows where deptno is not 10 or 20.

Ex8: Operator: <

Query: select * from student where rollno<2;

Result:

1|Ajay|Delhi|10

Ex9: Operator: <=

Query: `select * from student where deptno<=20;`

Result:

1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
9|Michael|Cochin|10
10|Pavan|Vijayawada|20
11|Suraj|Hyderabad|10
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Vizag|10

Ex10: Operator: >

Query:select * from student where deptno>30;

Result:
insert into student(rollno,name,city,deptno) values(16,'Navin','Bangalore',50);
select * from student where deptno>30;
5|Suneeta|Lucknow|40
8|Ali|Lucknow|40
12|Altaf|Bengaluru|40
16|Navin|Bangalore|50

We can see that result is only showing whose deptno is greater than or > 30. I.e. 40 and 50.
Note: I have added one more row with deptno 50.

Ex11: Operator: >=
Query: select * from student where deptno>=30;

Result:
4|Ramesh|Delhi|30
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
12|Altaf|Bengaluru|40
16|Navin|Bangalore|50

Greater than or equal to, it will include 30 also.

Ex12: Operator: =
Query: select * from student where city='Delhi';

Result:
1|Ajay|Delhi|10
4|Ramesh|Delhi|30
14|Verma|Delhi|20

So it is showing only those students whose city is Delhi.

Ex13: Operator: <>

Query: select * from student where city<>'Delhi';

Result:

2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijayawada|20
11|Suraj|Hyderabad|10
12|Altaf|Bengaluru|40
13|Ravi|Indore|20
15|Sharma|Vizag|10

Not equal to we write as <> . i.e. Greater than and less than symbol together. So, it is showing the results for those students who are not living in Delhi.

Ex14: Operator: AND

Query: select * from student where deptno>=30 AND city='Lucknow';

Result:

5|Suneeta|Lucknow|40
8|Ali|Lucknow|40

Ex15: Operator: OR

```
Query: select * from student where deptno>40 OR name='Navin';
```

```
Result:
16|Navin|Bengaluru|50
```

Ex16: Operator: not

Query: select * from student where deptno>=30 and NOT(city='Lucknow');

Result:
4|Ramesh|Delhi|30
6|Anita|Kolkata|30
7|Raj|Jaipur|30
12|Altaf|Bengaluru|40
16|Navin|Bangalore|50

Ex17: Operator: like

Query: select * from students where name like ‘A%’;

So, it will show only those students whose name starts with ‘A’ after that whatever may be there.
% means, any number of character after A.

Result:

1|Ajay|Delhi|10
3|Ajay|Mumbai|20
6|Anita|Kolkata|30
8|Ali|Lucknow|40
12|Altaf|Bengaluru|40

Query: **select * from students where name like ‘%y’;**

So, it will show only those students whose name ends with ‘y’ before that any number of character may be.

1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20

Query: **select * from students where name like ‘%m%’;**

It will show only those students who are having m in their name. We are saying any number of letter can be there(Even can’t be there), then m and then any number of letter can be there.

4	Ramesh	Delhi	30
9	Michael	Cochin	10
14	Verma	Delhi	20
15	Sharma	Vizag	10

order by:

Definition: The `ORDER BY` clause in SQL is used to sort the result set of a query in either ascending or descending order based on one or more columns. It can be used with `SELECT`, `UPDATE`, and `DELETE` statements to specify the order in which rows are returned or modified.

- **To print the result in ascending order, it is not mandatory to use `ASC` keyword.**

Ex1: select * from student order by name;

1|Ajay|Delhi|10
3|Ajay|Mumbai|20
8|Ali|Lucknow|40
12|Altaf|Bengaluru|40
6|Anita|Kolkata|30
9|Michael|Cochin|10
16|Navin|Bengaluru|50
10|Pavan|Vijayawada|20
17|Pintu|Delhi|60
18|Pintu|Delhi|
7|Raj|Jaipur|30
4|Ramesh|Delhi|30
13|Ravi|Indore|20
15|Sharma|Vizag|10
5|Suneeta|Lucknow|40
11|Suraj|Hyderabad|10
14|Verma|Delhi|20
2|Vijay|Kolkata|10

Ex2: select * from student order by name asc;
1|Ajay|Delhi|10
3|Ajay|Mumbai|20
8|Ali|Lucknow|40
12|Altaf|Bengaluru|40
6|Anita|Kolkata|30
9|Michael|Cochin|10
16|Navin|Bengaluru|50
10|Pavan|Vijayawada|20
17|Pintu|Delhi|60
18|Pintu|Delhi|
7|Raj|Jaipur|30
4|Ramesh|Delhi|30
13|Ravi|Indore|20
15|Sharma|Vizag|10
5|Suneeta|Lucknow|40
11|Suraj|Hyderabad|10
14|Verma|Delhi|20
2|Vijay|Kolkata|10

- **To print the result in descending order, it is mandatory to use `DESC` keyword.**

Ex3:select name from student order by name desc;
Vijay
Verma
Suraj
Suneeta
Sharma
Ravi
Ramesh
Raj
Pintu
Pintu
Pavan
Navin
Michael
Anita
Altaf

Ali
Ajay
Ajay

Ex4: select * from student order by name desc;
2|Vijay|Kolkata|10
14|Verma|Delhi|20
11|Suraj|Hyderabad|10
5|Suneeta|Lucknow|40
15|Sharma|Vizag|10
13|Ravi|Indore|20
4|Ramesh|Delhi|30
7|Raj|Jaipur|30
17|Pintu|Delhi|60
18|Pintu|Delhi|
10|Pavan|Vijayawada|20
16|Navin|Bengaluru|50
9|Michael|Cochin|10
6|Anita|Kolkata|30
12|Altaf|Bengaluru|40
8|Ali|Lucknow|40
1|Ajay|Delhi|10
3|Ajay|Mumbai|20

Cartesian Product:

Ex: If we join student and dept table, then each row of student table will be combined with other rows of the dept table.

Query: select * from student,dept;

When we write multiple table, then rows of these tables are multiplied and this is called as cartesian product.

Let's say we have 18 rows in student table and 6 rows in dept table. Then, the above query result will get us 108 rows.

In the above query we are combining student and dept table. Sample result will be as below where every row of student table is getting multiplied with every row of dept table.

1|Ajay|Delhi|10|10|CSE
1|Ajay|Delhi|10|20|ECE

1|Ajay|Delhi|10|30|Civil
1|Ajay|Delhi|10|40|Mech
1|Ajay|Delhi|10|50|IT
1|Ajay|Delhi|10|60|
2|Vijay|Kolkata|10|10|CSE
2|Vijay|Kolkata|10|20|ECE
2|Vijay|Kolkata|10|30|Civil
2|Vijay|Kolkata|10|40|Mech
2|Vijay|Kolkata|10|50|IT
2|Vijay|Kolkata|10|60|
.
.
.
.

JOIN

Definition: a JOIN is used to combine rows from two or more tables based on a related column between them. Or, In a query if we are retrieving the data from multiple tables, then we have to join the table.
Query: select * from student,dept where student.deptno=dept.deptno;
Result:

1|Ajay|Delhi|10|10|CSE
2|Vijay|Kolkata|10|10|CSE
3|Ajay|Mumbai|20|20|ECE
4|Ramesh|Delhi|30|30|Civil
5|Suneeta|Lucknow|40|40|Mech
6|Anita|Kolkata|30|30|Civil
7|Raj|Jaipur|30|30|Civil
8|Ali|Lucknow|40|40|Mech
9|Michael|Cochin|10|10|CSE
10|Pavan|Vijayawada|20|20|ECE
11|Suraj|Hyderabad|10|10|CSE
12|Altaf|Bengaluru|40|40|Mech
13|Ravi|Indore|20|20|ECE
14|Verma|Delhi|20|20|ECE
15|Sharma|Vizag|10|10|CSE
16|Navin|Bengaluru|50|50|IT
17|Pintu|Delhi|60|60|

As we can see above results, It will combine with the row, which is having matching primary key and foreign key. That is called as joining.

Now, we will write the above query using JOIN clause.
Once we say JOIN we must mention ON which column they must join. So, it becomes mandatory for us to write ON.

Query: select * from student join dept on student.deptno=dept.deptno;

Result:

1|Ajay|Delhi|10|10|CSE
2|Vijay|Kolkata|10|10|CSE
3|Ajay|Mumbai|20|20|ECE
4|Ramesh|Delhi|30|30|Civil
5|Suneeta|Lucknow|40|40|Mech
6|Anita|Kolkata|30|30|Civil
7|Raj|Jaipur|30|30|Civil
8|Ali|Lucknow|40|40|Mech
9|Michael|Cochin|10|10|CSE
10|Pavan|Vijayawada|20|20|ECE
11|Suraj|Hyderabad|10|10|CSE
12|Altaf|Bengaluru|40|40|Mech
13|Ravi|Indore|20|20|ECE
14|Verma|Delhi|20|20|ECE
15|Sharma|Vizag|10|10|CSE
16|Navin|Bengaluru|50|50|IT
17|Pintu|Delhi|60|60|

ALIAS or Renaming: **The ALIAS or renaming concept in SQL joins is used to assign a temporary name to a table or column in a query.**

Advantages of Alias:

- **Improve Readability and Simplify Queries.**
- **Handle Column Name Ambiguity.**
- **Combine Data from Multiple Tables with Clarity.**
- **Avoid Repetition of Table Names.**

Query: select * from student s join dept d on s.deptno = d.deptno;

Result:

1|Ajay|Delhi|10|10|CSE
2|Vijay|Kolkata|10|10|CSE
3|Ajay|Mumbai|20|20|ECE
4|Ramesh|Delhi|30|30|Civil
5|Suneeta|Lucknow|40|40|Mech
6|Anita|Kolkata|30|30|Civil
7|Raj|Jaipur|30|30|Civil
8|Ali|Lucknow|40|40|Mech
9|Michael|Cochin|10|10|CSE
10|Pavan|Vijayawada|20|20|ECE
11|Suraj|Hyderabad|10|10|CSE
12|Altaf|Bengaluru|40|40|Mech
13|Ravi|Indore|20|20|ECE
14|Verma|Delhi|20|20|ECE
15|Sharma|Vizag|10|10|CSE
16|Navin|Bengaluru|50|50|IT
17|Pintu|Delhi|60|60|

1. INNER JOIN

- **Definition: Returns only rows where there is a matching `deptno` between `student` and `dept`.**

Query: select student.rollno,student.name as sName,dept.dname from student inner join dept on student.deptno=dept.deptno;

Result:

1|Ajay|CSE
2|Vijay|CSE
3|Ajay|ECE
4|Ramesh|Civil
5|Suneeta|Mech
6|Anita|Civil
7|Raj|Civil
8|Ali|Mech
9|Michael|CSE
10|Pavan|ECE
11|Suraj|CSE
12|Altaf|Mech
13|Ravi|ECE
14|Verma|ECE
15|Sharma|CSE
16|Navin|IT

- **If we are renaming the table name, then we should use the temporary name to access column name.**

Query: select s.name as sName,d.dname as deptName from student as s inner join dept as d on s.deptno = d.deptno;

Result:

Ajay|CSE
Vijay|CSE
Ajay|ECE
Ramesh|Civil

Suneeta|Mech
Anita|Civil
Raj|Civil
Ali|Mech
Michael|CSE
Pavan|ECE
Suraj|CSE
Altaf|Mech
Ravi|ECE
Verma|ECE
Sharma|CSE
Navin|IT
Pintu|

- **Query will work with and without ‘as’ in same manner. As we can see in above query we have used ‘as’ and here we are not using. But results are same.**

Query: select s.name sName,d.dname from student s inner join dept d on s.deptno = d.deptno;

Result:

Ajay|CSE
Vijay|CSE
Ajay|ECE
Ramesh|Civil
Suneeta|Mech
Anita|Civil
Raj|Civil
Ali|Mech
Michael|CSE
Pavan|ECE
Suraj|CSE
Altaf|Mech
Ravi|ECE

Verma|ECE
Sharma|CSE
Navin|IT
Pintu|

2. LEFT JOIN:

- Returns all rows from the left table, and the matched rows from the right table.
- If there's no match, columns from the right table are filled with NULL.

Returns all rows from the `Students` table and matching rows from the `Dept` table. Rows without a match will have `NULL` for department details.

Explanation: If there is no match in the right table for a row in the left table, the result includes the row from the left table and fills the columns from the right table with `NULL`.

Inserted two records in student table as below to understand LEFT JOIN.

```
insert into student(rollno,name,city,deptno) values(17,'Pintu','Delhi',60);
insert into student(rollno,name,city,deptno) values(18,'Pintu','Delhi',NULL);
```

Query:

Ex1: select student.rollno,student.name as sName,dept.dname from student left join dept on student.deptno=dept.deptno;

1|Ajay|CSE
2|Vijay|CSE
3|Ajay|ECE
4|Ramesh|Civil
5|Suneeta|Mech
6|Anita|Civil
7|Raj|Civil

8|Ali|Mech
9|Michael|CSE
10|Pavan|ECE
11|Suraj|CSE
12|Altaf|Mech
13|Ravi|ECE
14|Verma|ECE
15|Sharma|CSE
16|Navin|IT
17|Pintu|
18|Pintu|

Ex2: select student.rollno,student.name as sName,dept.dname from student left join dept on student.deptno=dept.deptno where dept.dname is NULL;

17|Pintu|
18|Pintu|

3. RIGHT JOIN

- Returns **all rows from the right table**, and the matched rows from the left table.
- If there's no match, columns from the left table are filled with `NULL`.

- **Definition: Returns all rows from the `Dept` table and matching rows from the `Students` table. Rows without a match will have `NULL` for student details.**

Query:
select s.name as sName,d.dname as deptName from student as s right join dept as d on s.deptno = d.deptno;

Ajay|CSE
Vijay|CSE
Ajay|ECE
Ramesh|Civil

Suneeta|Mech
Anita|Civil
Raj|Civil
Ali|Mech
Michael|CSE
Pavan|ECE
Suraj|CSE
Altaf|Mech
Ravi|ECE
Verma|ECE
Sharma|CSE
Navin|IT
Pintu|

group by: We can group the rows by using one of the columns.

Group by column name should be same for select column name.

As we can see all the cities are showing only once.

Query: select city from student group by city;

Result:

Bengaluru
Cochin
Delhi
Hyderabad
Indore
Jaipur
Kolkata
Lucknow
Mumbai
Vijayawada
Vizag

group by with aggregate function

Query: select count(*),city from student group by city;

2|Bengaluru
1|Cochin
5|Delhi
1|Hyderabad
1|Indore
1|Jaipur

2|Kolkata
2|Lucknow
1|Mumbai
1|Vijayawada
1|Vizag

having Clause: If we want to impose some condition on group, we can use having clause.

Ex1: Show only those cities who are having 2 or more records.
Query: select count(*),city from student group by(city) having count(*)>=2;

Result:
2|Bengaluru
5|Delhi
2|Kolkata
2|Lucknow

Ex2: if you want to know which departments have more than 2 students.
Query: select deptno,count(name) studentcount from student group by deptno having count(*)>2;

Result:
10|5
20|4
30|3
40|3

Why Do We Need HAVING?

- **If we want to filter data before grouping, use the WHERE clause.**

- **But if we want to filter data after grouping, use the `HAVING` clause.**

Key Differences: WHERE vs HAVING

Feature	WHERE Clause	HAVING Clause
Use	Filters rows before aggregation.	Filters groups after aggregation.
Works With	Any column in the table.	Aggregate functions like `SUM`, `COUNT`, etc.
Example	`WHERE salary > 5000`	`HAVING AVG(salary) > 5000`

Aggregate Functions:

- **These are used with select clause and we can also use group by clause. So, they will work on groups.**
- **If we don't use group by then they will work entire relations.**

1. **count**

Ex1: select count(*) from student;
Result: 18

Ex2: select count(rollno) from student;
Result: 18

Ex3: select count(name) from student;
Result: 18

Ex4: select count(city) from student;
Result: 18

Ex5: select count(deptno) from student;
Result: 17
Here deptno is 17 as one deptno is NULL.

Ex6: select count(rollno),city from student group by city;
Result:
2|Bengaluru

1|Cochin
5|Delhi
1|Hyderabad
1|Indore
1|Jaipur
2|Kolkata
2|Lucknow
1|Mumbai
1|Vijayawada
1|Vizag

WIth this we will get no. of students in city.

2. **max**

Ex1: select max(rollno) from student;
Result: 18
Similarly, If we have marks of the student, we can find maximum marks of the student.

If we have prices of items, we can find max price.
Ex2: select max(rollno),city from student group by city;
16|Bengaluru
9|Cochin
18|Delhi
11|Hyderabad
13|Indore
7|Jaipur
6|Kolkata
8|Lucknow
3|Mumbai
10|Vijayawada
15|Vizag

There are few students in Bengaluru and the largest rollno is 16. Similarly, there are few students in Delhi and the largest roll no is 18 etc.

3. **min**

Ex: select min(rollno) from student;
Result: 1

4. **sum**

Ex1: select sum(rollno) from student;
Result: 171

Ex2: select sum(rollno),city from student group by city;
28|Bengaluru
9|Cochin
54|Delhi
11|Hyderabad
13|Indore
7|Jaipur
8|Kolkata
13|Lucknow
3|Mumbai
10|Vijayawada
15|Vizag

Ex3: select sum(marks),student from student;
We can find the sum of marks of a student in different subjects as shown above.

5. **avg**

Ex: select avg(rollno) from student;

Result: 9.5

Set Operations

These Set operations are same as Set operations in Mathematics.

1. **union : Means combining**

To understand this, we will fetch all the students from Delhi and Kolkata and then we will combine them to get the UNION result.

```
select * from student where city='Delhi';
1|Ajay|Delhi|10
4|Ramesh|Delhi|30
14|Verma|Delhi|20
17|Pintu|Delhi|60
18|Pintu|Delhi|

select * from student where city='Kolkata';
2|Vijay|Kolkata|10
6|Anita|Kolkata|30

select * from student where city='Delhi' union select * from student
where city='Kolkata';
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
4|Ramesh|Delhi|30
6|Anita|Kolkata|30
14|Verma|Delhi|20
17|Pintu|Delhi|60
18|Pintu|Delhi|
```

Note:

1. **It's mandatory to have same number of columns and same type or else we will get the error as below.**

select name from student where city='Delhi' union select rollno,name from student where city='Mumbai';
Parse error: SELECTs to the left and right of UNION do not have the same number of result columns

2. **In below examples, We have one Ajay in Delhi and one Ajay in Mumbai.**

Yes, UNION means it will combine two sets and if anything is common, then it will take only one copy.

Ex: select name from student where city='Delhi' union select name from student where city='Mumbai';
Result:
Ajay
Pintu
Ramesh
Verma

2. Intersect: will take only common

Ex: select name from student where city='Delhi' intersect select name from student where city='Mumbai';
Result: Ajay

3. **except: except common it will take remaining**

Ex: select name from student where city='Delhi' except select name from student where city='Mumbai';
Result:
Pintu
Ramesh
Verma

Subquery: First Inner query will be executed then outer query.

Statement: Find all those students, who are living in the city where Ajay is living.

Note: If we have inner query result more than one, use ‘in’
Ex: select * from student where city in(select city from student where name='Ajay');

Result:
1|Ajay|Delhi|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
14|Verma|Delhi|20
17|Pintu|Delhi|60
18|Pintu|Delhi|

Statement: Find all the students whose deptno is same as ‘Ajay’
If we have inner query result only one, use ‘=’
Ex: select * from student where deptno = (select deptno from student where name='Ajay');

Result:
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
9|Michael|Cochin|10
11|Suraj|Hyderabad|10
15|Sharma|Vizag|10

Statement: Find all the students whose rollno is greater than ‘Suraj’
Ex: select * from student where rollno > (select rollno from student where name='Suraj');

Result:
12|Altaf|Bengaluru|40
13|Ravi|Indore|20

14|Verma|Delhi|20
15|Sharma|Vizag|10
16|Navin|Bengaluru|50
17|Pintu|Delhi|60
18|Pintu|Delhi|

Ex: select * from student where rollno > (select avg(rollno) from student);
Result:
10|Pavan|Vijayawada|20
11|Suraj|Hyderabad|10
12|Altaf|Bengaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Vizag|10
16|Navin|Bengaluru|50
17|Pintu|Delhi|60
18|Pintu|Delhi|