MySQL Class Notes

What are Data and Information?

Data is facts and statistics stored or flow over a network, generally, it's raw and unprocessed. It is an individual unit that contains raw materials.

Information is processed, organized, and structured data that has meaning and context. It is data that has been interpreted in a way that makes it useful for decision-making or communication.

Example: Raw marks scored by students are **data**, but when you calculate averages, ranks, or trends, that becomes **information**.

What is Database?

By definition, a database is a collection of structured and related data organized in a way that data can be easily accessed and managed.

In short, the database is one way to store data. You deal with databases every day, for example, your photos are stored in your smartphone's gallery, this gallery is a database.

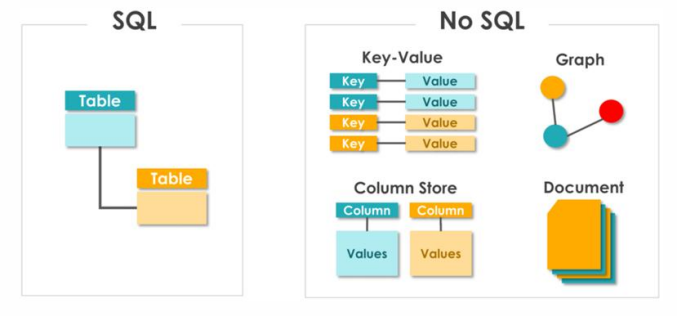
There are many different types of databases such as

Relational databases:

A Relational (SQL) Database organizes data into one or more tables (or "relations") with a fixed schema. This means that each table has a predefined structure of columns and rows, and the data within it conforms to specific data types and constraints. The tables are linked together through common columns, known as primary and foreign keys, to establish relationships.

Non-Relational Database (NoSQL)

A Non-Relational (NoSQL) Database provides a flexible way to store and retrieve data that doesn't fit into the traditional tabular structure. These databases are designed to be highly scalable and can handle unstructured or semi-structured data. They do not enforce a fixed schema, allowing for much more agile development and the ability to store a wide variety of data types



What is DBMS?

A DBMS is software that helps you **store, organize, and manage data** in a database.

It allows you to easily **create, update, delete, and retrieve** data using a structured way.

It provides an interface and a set of tools to perform various operations, such as:

- Creating databases and tables
- Querying data
- Updating and managing data

Examples::XML, NoSQL databases (e.g., MongoDB, Cassandra), older flat-file systems.

What is RDBMS ?

An RDBMS is a type of DBMS that stores data in the form of **tables (rows and columns)** and maintains **relationships** between them.

It follows the concept of **relations**, where data from different tables can be linked using **keys**.

Example: Think of a **college system**.

1. One table stores **student details**.
2. Another table stores **course details**.
3. A third table stores **marks**.
4. By using **relationships (keys)**, you can connect which student belongs to which course and what marks they scored.

EX: MySQL, PostgreSQL, Oracle, SQL Server.

DBMS vs **RDBMS** comparison table

Feature	DBMS	RDBMS
Data Storage	Stores data as files or simple forms	Stores data in **tables (rows & columns)**
Relationships	No relationships between data	Supports **relationships** using **primary key & foreign key**
Normalization	Does not support normalization	Supports normalization to reduce redundancy
Data Integrity	Low data integrity	High data integrity due to constraints
Complexity	Simple to use, suitable for small data	More complex, suitable for large data
Examples	dBase, MS Access	MySQL, Oracle, SQL Server, PostgreSQL

➔ **DBMS** = simple storage + no relations

➔**RDBMS** = table-based + relationships

What is SQL?

SQL (Structured Query Language):

SQL is a **standard programming language** used to communicate with databases.

It helps you to **store, retrieve, update, and delete** data in a structured way.

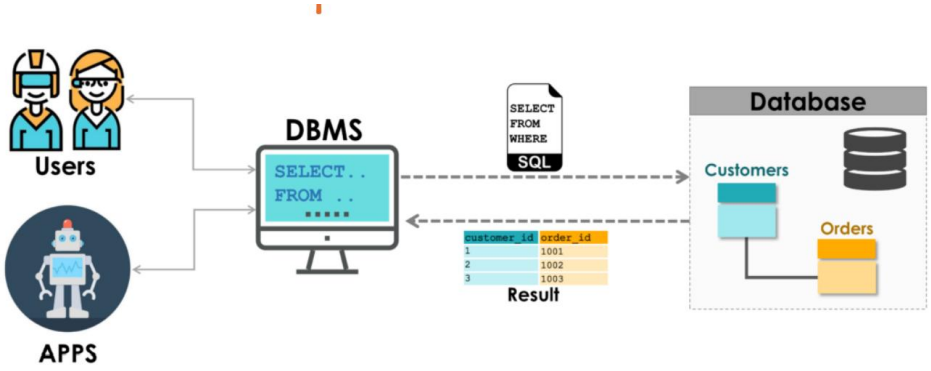
How does SQL Works?

SQL acts as a **communication language** between the user and the database.

1. The **user writes SQL commands** (like SELECT, INSERT, UPDATE, DELETE).
2. The **DBMS/RDBMS interprets** these commands.
3. The system then **executes the query** and returns the required result or updates the data.

Example:

1. You type: SELECT * FROM students WHERE grade = 'A';
2. SQL passes this request to the database.
3. The database searches the table and gives you all students with grade A.



Why We Need to Learn SQL:

1. **Widely Used:** SQL is one of the most-used languages for working with databases.
2. **High Demand:** Developers with SQL skills are always in demand across industries.
3. **Universal Application:** SQL is used everywhere — from small businesses to big tech companies.
4. **Easy to Learn:** Compared to other programming languages, SQL is simple and beginner-friendly.
5. **Career Growth:** SQL provides one of the most impactful skills for IT professionals, making it a smart investment for career improvement.

History of SQL

Introduction

1. SQL stands for **Structured Query Language**.
2. It is the standard language used to store, manipulate, and retrieve data from relational databases.
3. SQL is used in almost all RDBMS (Relational Database Management Systems) like Oracle, MySQL, SQL Server, PostgreSQL, and SQLite.

Origin

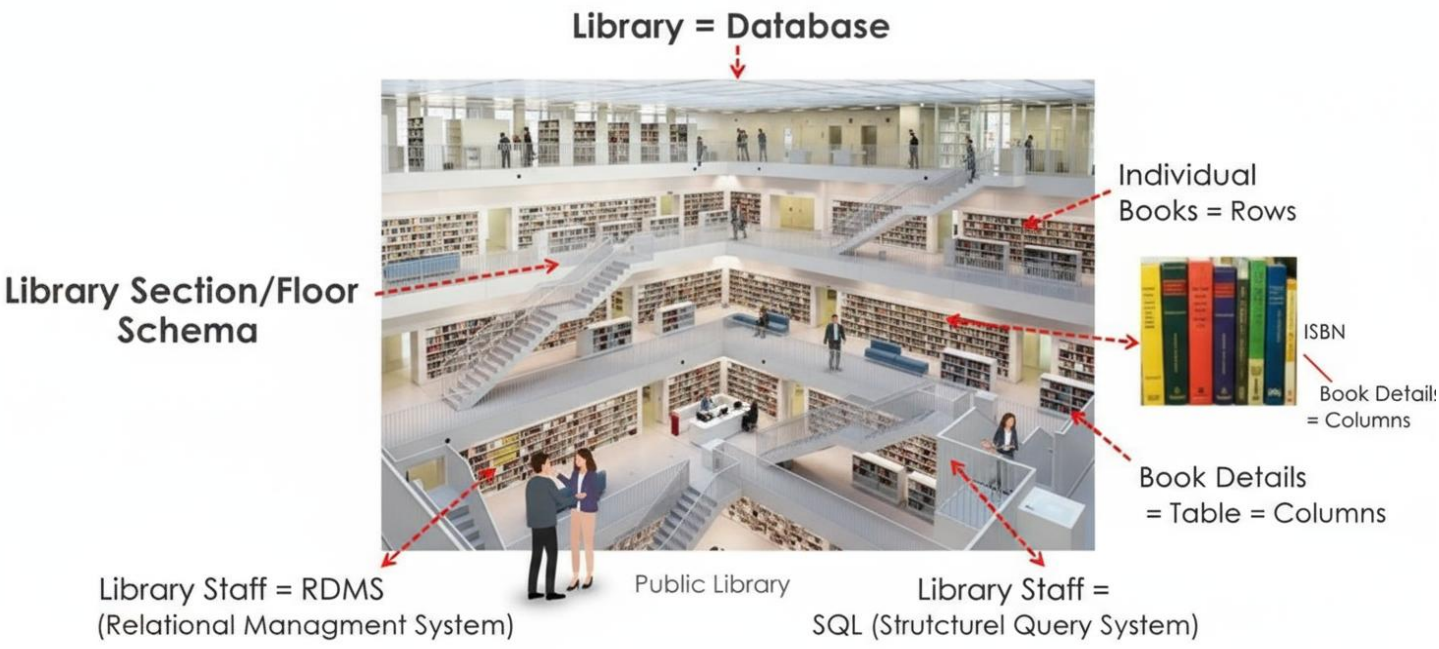
1. In the **1970s**, **IBM** researchers **Donald D. Chamberlin** and **Raymond F. Boyce** developed a language called **SEQUEL** (*Structured English Query Language*).
2. It was designed to interact with IBM's **System R** — the first prototype of a **Relational Database Management System (RDBMS)** based on **E. F. Codd's relational model**.

Evolution Timeline

Year	Event
1970	E. F. Codd, an IBM researcher, published a paper titled *"A Relational Model of Data for Large Shared Data Banks"*, introducing the **relational model**.
1974	**SEQUEL** was developed at IBM to query relational databases.
1977	SEQUEL's name was changed to **SQL** (due to trademark issues).

1979	**Oracle** became the **first company** to release a **commercial SQL-based RDBMS**.
1986	**ANSI** (American National Standards Institute) adopted SQL as the **standard database language**.
1987	**ISO** (International Organization for Standardization) also approved SQL as the **international standard**.
1989–1992	SQL evolved with new standards: **SQL-89**, **SQL-92**, and later **SQL:1999** introducing new features like triggers, object-relational concepts, etc.

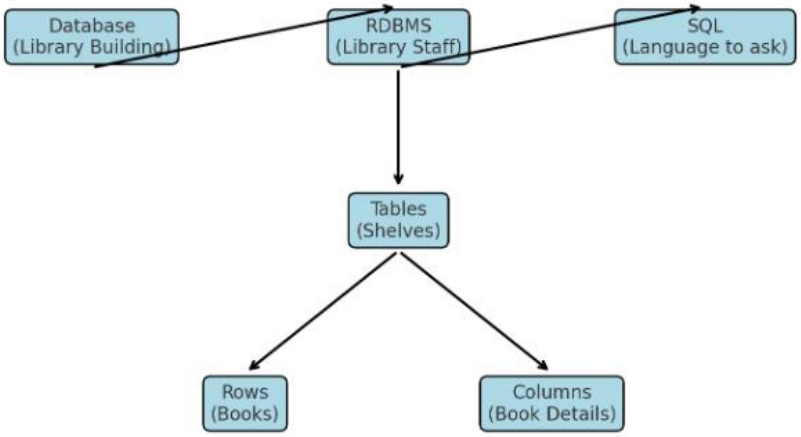
How is SQL Database organized?



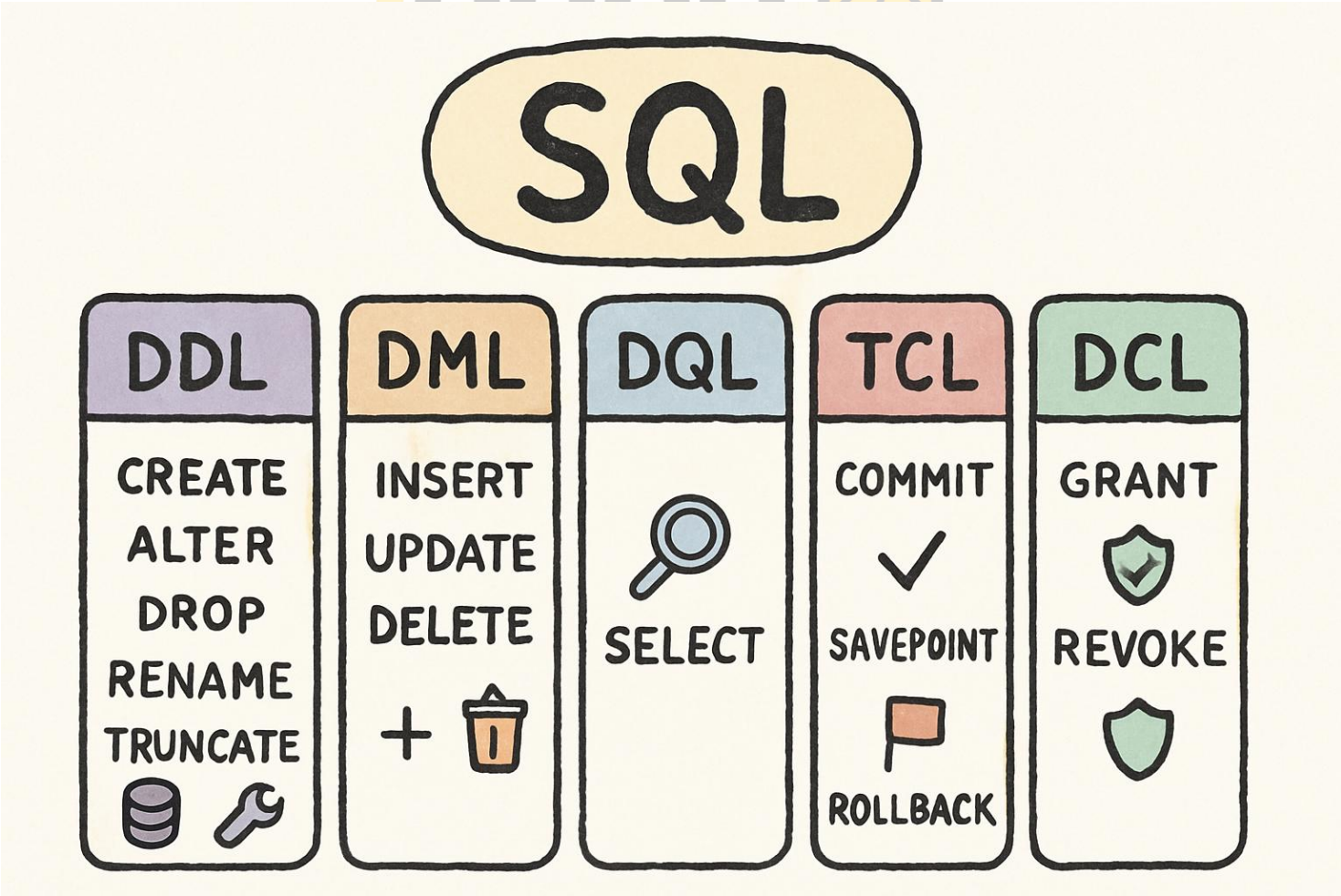
Library Analogy for Databases

1. Library Analogy for Databases
2. Database → The library building (the place where all data/books are stored).
3. RDBMS → The library's management system and staff (they organize, manage, and keep track of the books/data).
4. SQL → The language we use to communicate with the library staff (ask for what we want).
5. Tables → The shelves in the library (where books are arranged).
6. Rows → The individual books on a shelf (each record of data).
7. Columns → The book details like title, author, or ISBN (attributes of the data).

RDBMS stores and organizes the data, while SQL is how we ask for it.



SQL COMMANDS



SQL Commands

DDL (Data Definition Language)

1. CREATE → used to create database objects (table, view, index, etc.)
2. ALTER → used to modify an existing table (add/modify/drop columns)
3. DROP → used to delete a database object permanently
4. TRUNCATE → used to remove all records from a table (structure remains)
5. RENAME → used to change the table name

DML (Data Manipulation Language)

1. INSERT → used to add new records into a table
2. UPDATE → used to modify existing records in a table
3. DELETE → used to remove records from a table

DQL (Data Query Language)

1. SELECT → used to fetch data from a table

DCL (Data Control Language)

1. GRANT → used to give permissions to users
2. REVOKE → used to take back permissions from users

TCL (Transaction Control Language)

1. COMMIT → used to save the changes permanently
2. ROLLBACK→ used to undo changes
3. SAVEPOINT → used to set a point in a transaction to roll back to later

SQL Data Types

Numeric Data Types

MySQL supports the SQL standard integer types INTEGER (or INT) and SMALLINT. As an extension to the standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each integer type.

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	−263-2^{63}−263	0	263−12^{63}-1263−1	264−12^{64}-1264−1

String datatype:

String datatypes are data types used to store sequences of characters such as letters, numbers, and symbols.
In simple words, they represent textual data.

Type	Storage (Bytes)	Minimum Length	Maximum Length	Notes/Use Case
CHAR(n)	n	1	255 chars	Fixed-length, right-padded
VARCHAR(n)	n + 1 or 2 (length prefix)	0	65,535 chars*	Variable-length, efficient for names
TEXT	Varies: 2 + string size	0	65,535 chars (M	For large/unstructured text
ENUM	gender ENUM('Male', 'Female', 'Other'),	1	255 allowed values	Restricted set, stored as integer

Date Datatypes in MySQL

If we want to store dates or time in a table, we use date datatypes.

Type	Storage (Bytes)	Range**	Format/Notes
DATE	3	1000-01-01 to 9999-12-31	'YYYY-MM-DD'
DATETIME	8	1000-01-01 00:00:00 to 9999-12-31 23:59:59	'YYYY-MM-DD hh:mm:ss'
TIMESTAMP	4	1970-01-01 00:00:01 to 2038-01-19 03:14:07	Unix timestamp, UTC, auto-updates
TIME	3	-838:59:59 to 838:59:59	'hh:mm:ss', time interval support
YEAR	1	1901 to 2155	4-digit year only

➔Till now, we learned about datatypes (String, Date, etc.).

➔ Now, we need to know where to store data → inside Databases and Tables.

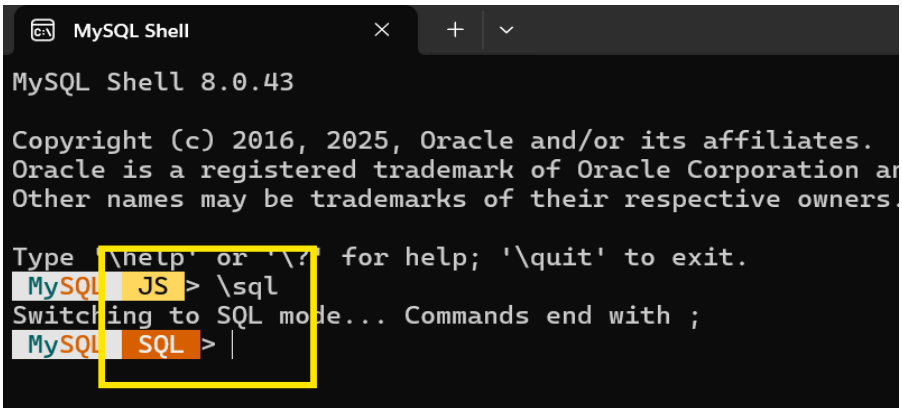
Database: A container that holds tables.

Table: Structure inside a database where actual data is stored (rows & columns).

Connecting to MySQL Shell

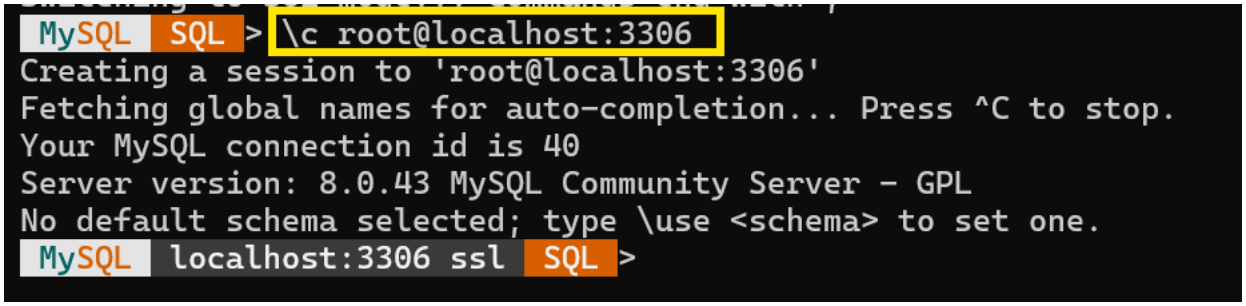
1. Change to JS Mode TO SQL:

   Run the command: \sql



Connect to MySQL Server:

Use the connect command:\c root@localhost:3306



Enter Password:

After running the connect command, you'll be prompted to enter the password for the user (root in this case). Enter the password you set during MySQL installation for the root user.

Command	Description
\sql	Switch to JS mode TO SQL in MySQL Shell
\connect root@localhost:3306 or \c root@localhost:3306	Connect to MySQL server as root on localhost port 3306
Enter password when prompted	Authenticate user

Once connected to MySQL Shell, here are the next common steps you can do:

1. Create a Database

Syntax:CREATE DATABASE database_name;

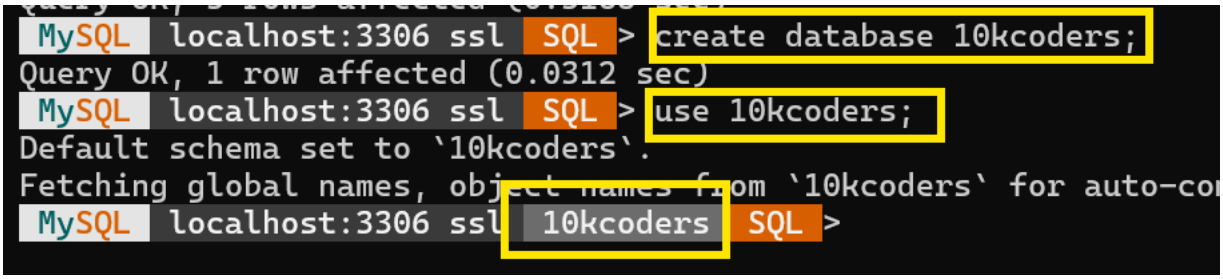
Example:CREATE DATABASE 10kcoders;

After creating the database, we need to use this database to create a table inside.

2. Use a Database

Syntax: USE database_name;

Example: USE 10kcoders;



DDL (Data Definition Language)

Create is used to create database and create table and to create a user::

1. CREATE

Description: Creates a new table, database, index, or other database objects.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

Example::

```
CREATE TABLE employees (
    ID INT,
    Name VARCHAR(50),
    DOB DATE,
    NAN INT,
    Location VARCHAR(255)
);
```

It will create a table called employees to **see the structure** we have a key word called

DESC or **DESCRIBE** command in MySQL is to show the structure of a table. It provides detailed information about each column of the table, including:

1. Field: The name of the column.
2. Type: The data type of the column (e.g., INT, VARCHAR).
3. Null: Whether the column can contain NULL values or not.
4. Key: Whether the column is indexed (e.g., PRIMARY KEY).
5. Default: The default value for the column, if any.
6. Extra: Additional information such as auto-increment.

SYNTAX:: **DESC EMPLOYEES;**

```
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| ID       | int          | YES  |     | NULL    |       |
| Name     | varchar(50)  | YES  |     | NULL    |       |
| DOB      | date         | YES  |     | NULL    |       |
| NIN      | int          | YES  |     | NULL    |       |
| Location | varchar(255) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

SHOW DATABASES

1. Purpose: Lists all available databases in the MySQL server to which you have access.
2. Syntax: **SHOW DATABASES;**

```
MySQL localhost:3306 ssl 10kcoders SQL > SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| 10kcoders          |
| d3r                |
| d4r                |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

SHOW TABLES

1. Purpose: Lists all tables within the currently selected database.
2. Syntax: **SHOW TABLES;**

```
7 rows in set (0.0017 sec)
MySQL localhost:3306 ssl 10kcoders SQL > SHOW TABLES;
+--------------------+
| Tables_in_10kcoders |
+--------------------+
| employees          |
+--------------------+
1 row in set (0.3163 sec)
MySQL localhost:3306 ssl 10kcoders SQL >
```

ALTER COMMAND

The ALTER command is used to change the structure of an existing table We can:

1)Add column

Add a Column at the Beginning (First Position)

Syntax: Alter table table_name add column column _name data_type first;

Ex: Alter table employees add column empid int first;

```
1 row in set (0.3163 sec)
MySQL localhost:3306 ssl 10kcoders SQL > alter table employees add column empid int first;
Query OK, 0 rows affected (0.3671 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| empid    | int          | YES  |     | NULL    |       |
| ID       | int          | YES  |     | NULL    |       |
| Name     | varchar(50)  | YES  |     | NULL    |       |
| DOB      | date         | YES  |     | NULL    |       |
```

2)Add a Column in the Middle (After a Specific Column)

Syntax : ALTER TABLE my_table ADD COLUMN new_column INT AFTER existing_column;

Ex : : Alter table employees add column salary int after dob;

```
6 rows in set (0.0456 sec)
MySQL localhost:3306 ssl 10kcoders SQL > alter table employees add column salary int  after dob;
Query OK, 0 rows affected (0.0967 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc emp;
ERROR: 1146 (42S02): Table '10kcoders.emp' doesn't exist
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| empid    | int          | YES  |     | NULL    |       |
| ID       | int          | YES  |     | NULL    |       |
| Name     | varchar(50)  | YES  |     | NULL    |       |
| DOB      | date         | YES  |     | NULL    |       |
| salary   | int          | YES  |     | NULL    |       |
| NIN      | int          | YES  |     | NULL    |       |
| Location | varchar(255) | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

3)Add a Column at the End (Last Position)

Syntax : : Alter table table_name add column column_name datatype(SIZE);

Ex: Alter table employees add column deptno int;

```
MySQL localhost:3306 ssl 10kcoders SQL > alter table employees add column deptno int;
Query OK, 0 rows affected (0.0933 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| empid    | int          | YES  |     | NULL    |       |
| ID       | int          | YES  |     | NULL    |       |
| Name     | varchar(50)  | YES  |     | NULL    |       |
| DOB      | date         | YES  |     | NULL    |       |
| salary   | int          | YES  |     | NULL    |       |
| NIN      | int          | YES  |     | NULL    |       |
| Location | varchar(255) | YES  |     | NULL    |       |
| deptno   | int          | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

Rename column:

Syntax : :Alter table table_name rename column old_name to new_column_name ;

Ex : Alter table employees rename column NIN to commission;

```
MySQL localhost:3306 ssl 10kcoders SQL > Alter table employees rename column NIN to commission;
Query OK, 0 rows affected (0.0707 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| empid      | int          | YES  |     | NULL    |       |
| ID         | int          | YES  |     | NULL    |       |
| Name       | varchar(50)  | YES  |     | NULL    |       |
| DOB        | date         | YES  |     | NULL    |       |
| salary     | int          | YES  |     | NULL    |       |
| commission | int          | YES  |     | NULL    |       |
| Location   | varchar(255) | YES  |     | NULL    |       |
| deptno     | int          | YES  |     | NULL    |       |
```

To Modify datatype::

Syntax : Alter table table_name modify column _name datatype(size)

EX : :ALTER TABLE employees MODIFY salary DECIMAL(10,2);

```
MySQL localhost:3306 ssl 10kcoders SQL > ALTER TABLE employees MODIFY salary DECIMAL(10,2);
Query OK, 0 rows affected (0.1079 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| Name       | varchar(50)   | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| Location   | varchar(255)  | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

To increase the data type size

Syntax : Alter table table_name modify column _name datatype(newsize)

EX : :ALTER TABLE employees MODIFY name varchar(500);

```
MySQL localhost:3306 ssl 10kcoders SQL > ALTER TABLE employees MODIFY name varchar(500);
Query OK, 0 rows affected (0.0978 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl 10kcoders SQL > desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| name       | varchar(500)  | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| Location   | varchar(255)  | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
8 rows in set (0.0035 sec)
```

To change data type to int to varchar()

Syntax : Alter table table_name modify column _name new datatype(newsize);

EX : :ALTER TABLE employees MODIFY name int;

Drop column

Syntax : Alter table table_name drop column column _name ;

EX : ALTER TABLE employees drop location;

```
8 rows in set (0.0035 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > ALTER TABLE employees drop location;
Query OK, 0 rows affected (0.1038 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:33060+ ssl 10kcoders SQL > desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| name       | varchar(500)  | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

Change column name and datatype

Syntax:: Alter table student change column old_column new column datatype(size);

Ex:: ALTER TABLE emp CHANGE column name ename varchar(50);

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > ALTER TABLE employees CHANGE column name ename varchar(50);
Query OK, 0 rows affected (0.0817 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > desc employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| ename      | varchar(50)   | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

To drop a column fromm table:

Syntax ::alter table table_name drop column column_name;

Ex: alter table emp drop column id;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > desc emp;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| ename      | varchar(50)   | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
7 rows in set (0.0033 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > alter table emp drop column id;
Query OK, 0 rows affected (0.0903 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > desc emp;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ename      | varchar(50)   | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

RENAME TABLE

What it does: Changes the name of an existing table.

Syntax: RENAME TABLE old_table_name TO new_table_name;

Ex:: rename table employees to emp;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > rename table employees to emp;
Query OK, 0 rows affected (0.1482 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > desc emp;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| empid      | int           | YES  |     | NULL    |       |
| ID         | int           | YES  |     | NULL    |       |
| ename      | varchar(50)   | YES  |     | NULL    |       |
| DOB        | date          | YES  |     | NULL    |       |
| salary     | decimal(10,2) | YES  |     | NULL    |       |
| commission | int           | YES  |     | NULL    |       |
| deptno     | int           | YES  |     | NULL    |       |
```

DROP TABLE

What it does: Permanently deletes a table and all its data. The table structure is removed and cannot be recovered without backup.

Syntax: DROP TABLE table_name;

Example: DROP TABLE employee;

TRUNCATE TABLE

What it does: Deletes all rows in a table but keeps the table structure for future use.

It is faster than DELETE without a WHERE clause.

Syntax:**TRUNCATE TABLE** table_name;

Example:**TRUNCATE TABLE** employee;

DML (Data Manipulation Language

1. Used to manipulate data within tables.

Commands:

1. **INSERT**: Adds new records to a table.
2. **UPDATE**: Modifies existing records.
3. **DELETE:** Removes records from a table.

Insert values into all columns (in order)

Syntax :: INSERT INTO table_name VALUES (value1, value2, value3);

Ex:: INSERT INTO emp VALUES (101, 'Rahul', 2021-02-21,3000.32);

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > INSERT INTO EMP VALUES(101,'RAM','2021-02-22',20000.32);
Query OK, 1 row affected (0.0306 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM EMP;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
+-------+-------+------------+----------+
1 row in set (0.0010 sec)
```

Explanation: Inserts data for all columns in the exact order they were created

Insert values into specific columns

Syntax:: INSERT INTO table_name (col1, col2) VALUES (val1, val2);

Ex:: INSERT INTO EMP (ename, salary) VALUES ('Priya', 2100.32);

```
1 row in set (0.0010 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > INSERT INTO EMP (ename, salary) VALUES ('Priya', 2100.32);
Query OK, 1 row affected (0.0287 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
|  NULL | Priya | NULL       |  2100.32 |
+-------+-------+------------+----------+
2 rows in set (0.0015 sec)
```

Explanation: Inserts data only into name and age columns, leaving id as NULL (or default value if defined)

Insert Multiple Rows (One Query)

Syntax:: INSERT INTOtable_name (col1, col2, col3) VALUES (val1a, val2a,val3a), (val1b, val2b, val3b), (val1c, val2c, val3c);

Ex::

INSERT INTO EMP VALUES

(104, 'Sita', '2022-05-10', 25000.50),

(105, 'Ravi', '2020-11-15', 30000.00),

(106, 'Kiran', '2021-08-01', 28000.75);

```
3 rows in set (0.0206 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > INSERT INTO EMP VALUES
                                          -> (104, 'Sita', '2022-05-10', 25000.50),
                                          -> (105, 'Ravi', '2020-11-15', 30000.00),
                                          -> (106, 'Kiran', '2021-08-01', 28000.75);
Query OK, 3 rows affected (0.0261 sec)

Records: 3  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM EMP;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
|  NULL | Priya | NULL       |  2100.32 |
|   103 | Amit  | 2023-03-22 | 40000.32 |
|   104 | Sita  | 2022-05-10 | 25000.50 |
|   105 | Ravi  | 2020-11-15 | 30000.00 |
|   106 | Kiran | 2021-08-01 | 28000.75 |
+-------+-------+------------+----------+
6 rows in set (0.0032 sec)
```

Explanation: Inserts 3 rows into the table in a single query

To update table

SYNTAX::UPDATE table_nameSET column1 = value1, column2 = value2 WHERE condition;

Ex: UPDATE student SET emp_id= 102 WHERE ename =Priya;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > UPDATE emp SET empid= 102 WHERE ename ='Priya';
Query OK, 1 row affected (0.0281 sec)

Rows matched: 1  Changed: 1  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
|   102 | Priya | NULL       |  2100.32 |
|   103 | Amit  | 2023-03-22 | 40000.32 |
|   104 | Sita  | 2022-05-10 | 25000.50 |
|   105 | Ravi  | 2020-11-15 | 30000.00 |
|   106 | Kiran | 2021-08-01 | 28000.75 |
+-------+-------+------------+----------+
```

Example (multiple columns)

UPDATE emp

SET dob = '2034-05-29', salary = 25000.25

WHERE empid = 102;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > UPDATE emp
                                          -> SET dob = '2034-05-29', salary = 25000.25
                                          -> WHERE empid = 102;
Query OK, 1 row affected (0.0293 sec)

Rows matched: 1  Changed: 1  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
|   102 | Priya | 2034-05-29 | 25000.25 |
|   103 | Amit  | 2023-03-22 | 40000.32 |
|   104 | Sita  | 2022-05-10 | 25000.50 |
|   105 | Ravi  | 2020-11-15 | 30000.00 |
|   106 | Kiran | 2021-08-01 | 28000.75 |
```

To DELETE single row in table

Syntax:: DELETE FROM table_name WHERE condition;

Example: DELETE FROM emp WHERE id = 106;

Explanation: Deletes the row where id = 106.

Example (delete all rows):

Ex:: DELETE FROM student;

Note: This removes all data but keeps the table structure

(similar to TRUNCATE, but slower and logs each deletion).

Constraints

What is a Constraint?

A constraint is a rule that we apply on table columns to restrict the type of data that can be stored.

1. It ensures accuracy, validity, and reliability of the data in a database.
2. Example: If we don't want duplicate employee IDs, we use a PRIMARY KEY constraint.

Why Constraints are Important?

1. Prevents invalid data (e.g., salary cannot be negative).
2. Avoids duplicate records.
3. Maintains relationships between tables.
4. Improves data integrity.

Types of Constraints in MySQL

1. **NOT NULL** → Ensures column cannot have NULL values.
2. **UNIQUE** → Ensures all values in a column are unique.
3. **PRIMARY KEY** → Combines NOT NULL + UNIQUE. Identifies each row uniquely.
4. **FOREIGN KEY** → Links two tables and maintains referential integrity.
5. **CHECK** → Ensures condition is true for values (e.g., salary > 0).
6. **DEFAULT** → Assigns a default value if none is provided.
7. **AUTO_INCREMENT** → is used to **automatically generate unique numbers** for a column when a new row is inserted.

Constraint Naming and Definition

The points about constraint definition and naming are true for all standard constraints in SQL (specifically MYSQL, as you mentioned it):

1. **Column Level:** Constraints like NOT NULL, UNIQUE, and PRIMARY KEY can be defined directly after the column's data type.
2. **Table Level:** All constraints can be defined after the column definitions but before the closing parenthesis of the CREATE TABLE statement.
3. **Alter Level:** Constraints can be added later using the ALTER TABLE command.

Naming Conventions

System-Generated Name:

If you define a constraint *without* the CONSTRAINT clause (e.g., ... column_name VARCHAR2(10) PRIMARY KEY), MYSQL automatically assigns a name like SYS_C00....

User-Defined Name:

If you want a descriptive name (which is good practice for easier management and error identification), you use the CONSTRAINT clause *before* the constraint type (e.g., ... CONSTRAINT pk_employee_id PRIMARY KEY (employee_id)).

MySQL Constraints

NOT NULL Constraint

1. Used to avoid NULL values.
2. Can be applied only at column level.

EX::
```
CREATE TABLE Student (
    ID  INT NOT NULL,
    Name VARCHAR(10),
    Marks INT
);
```

2. CHECK Constraint

1. Ensures that a condition is satisfied for values.
2. Can be applied at column level, table level, or alter level.

Examples:

(a) Column Level:

```
EX::CREATE TABLE Student (
  ID INT(2),
  Name VARCHAR(10),
  Marks INT CHECK (Marks > 300)
);
```

(b)Table Level:

```
CREATE TABLE Student (
  No INT(2),
  Name VARCHAR(10),
  Marks INT(3),
  CONSTRAINT Chk_Constraint CHECK (Marks > 300)
);
```

(c)Alter Level:

```
-- Without name
ALTER TABLE Student ADD CHECK (Marks > 300);
-- With name
ALTER TABLE Student ADD CONSTRAINT Chk_Constraint CHECK (Marks > 300);
```

To see the CHECK condition:

SYNTAX::SHOW CREATE TABLE Student;

3. UNIQUE Constraint

1. Ensures all values in a column are unique.
2. Allows NULL values (but only one).

Examples:

Column Level:

```
CREATE TABLE Student (
  No INT UNIQUE,
  Name VARCHAR(10)
);
```

Table Level:

```
CREATE TABLE Student (
  No INT,
  Name VARCHAR(10),
  CONSTRAINT Unq_Constraint UNIQUE (No)
);
```


Alter Level:

```
ALTER TABLE Student ADD UNIQUE (No);

-- With name

ALTER TABLE Student ADD CONSTRAINT Unq_Constraint UNIQUE (No);
```

4. DEFAULT Constraint

1. If you do **not provide a value**, SQL automatically inserts the default value.
2. If you **provide a value**, that value is taken instead of the default.

Examples:

(a) Column Level:

```
CREATE TABLE Student (
 No INT,
 Name VARCHAR(10),
 City VARCHAR(20) DEFAULT 'Hyderabad'
);
```

(b) Alter Level:

```
-- Add default
ALTER TABLE Student MODIFY City VARCHAR(22) DEFAULT 'Hyderabad';
-- Drop default
ALTER TABLE Student ALTER City DROP DEFAULT;
-- Another way
ALTER TABLE Student ALTER City SET DEFAULT 'Hyderabad';
```

5. PRIMARY KEY Constraint

1. Ensures each row is uniquely identified.
2. Combines NOT NULL + UNIQUE.

Examples:(a) Column Level:

```
CREATE TABLE Student (
 No INT PRIMARY KEY,
 Name VARCHAR(10) );
```

(b) Table Level:

```
CREATE TABLE Student (
  No INT,
  Name VARCHAR(10),
  CONSTRAINT PK_Student PRIMARY KEY (No)
);
```

c)Alter Level:

```
ALTER TABLE Student ADD PRIMARY KEY (No);
-- With name
ALTER TABLE Student ADD CONSTRAINT PK_Student PRIMARY KEY (No);
```

6. FOREIGN KEY Constraint

1. **Maintains referential integrity between two tables.**

Examples:

(a) Column Level:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50)
);
CREATE TABLE Course (
  CourseID INT PRIMARY KEY,
  StudentID INT REFERENCES Student(StudentID)
);
```

hear we have to join two tables student table and course table

(b) Table Level:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50)
);
CREATE TABLE Course (
  CourseID INT PRIMARY KEY,
  StudentID INT,
  CONSTRAINT FK_Course_Student FOREIGN KEY (StudentID)
  REFERENCES Student(StudentID)
);
```

c) Alter Level:

```
ALTER TABLE Course
ADD CONSTRAINT FK_Course_Student
FOREIGN KEY (StudentID) REFERENCES Student(StudentID);
```

Composite Keys

A Composite Key can be defined on a combination of columns.

Composite key can be defined in table and alter levels only.

Table LEVEL Example:

```
CREATE TABLE EMP(
 Empno Number(2),Ename Varchar(10),
Deptno Number(2),
 UNIQUE(Empno, Ename) );
```

Alter LEVEL Example:

SQL> ALTER TABLE EMP ADD UNIQUE (Empno, Ename);

(OR)

SQL> ALTER TABLE EMP ADD CONSTRAINT Comp_Key_Constraint UNIQUE(Empno, Ename);

Operators in MySQL

In MySQL, we can retrieve specific rows from a table using conditions.

Two important clauses are:

1. WHERE Clause

1. Used to filter records based on a condition.
2. Without WHERE, all rows are returned.

2. ORDER BY Clause

1. Used to sort the records in ascending (ASC) or descending (DESC) order.
2. By default, sorting is in ascending order.

1. Arithmetic Operators

Used for mathematical calculations.

Operator	Description	Example
+	Addition	SELECT 10 + 5; → 15
-	Subtraction	SELECT 10 - 5; → 5
*	Multiplication	SELECT 10 * 5; → 50
/	Division	SELECT 10 / 5; → 2
%	Modulus (remainder)	SELECT 10 % 3; → 1

2. Comparison Operators

Used to compare two values, mostly in WHERE clauses

Operator	Description	Example
=	Equal to	SELECT * FROM emp WHERE salary = 5000;
!= or <>	Not equal to	SELECT * FROM emp WHERE deptno <> 10;
>	Greater than	SELECT * FROM emp WHERE salary > 3000;
<	Less than	SELECT * FROM emp WHERE age < 25;
>=	Greater than or equal to	SELECT * FROM emp WHERE salary >= 10000;
<=	Less than or equal to	SELECT * FROM emp WHERE age <= 30;
BETWEEN … AND …	Between a range	salary BETWEEN 5000 AND 10000
IN (…)	Matches any value in list	deptno IN (10,20,30)
LIKE	Pattern matching with wildcards (%, _)	ename LIKE 'S%'
IS NULL	Checks if value is NULL	comm IS NULL

3. Logical Operators

Used to combine multiple conditions.

Operator	Description	Example
AND	True if all conditions are true	WHERE deptno=10 AND salary>5000
OR	True if any condition is true	WHERE deptno=10 OR deptno=20
NOT	Negates condition	WHERE NOT deptno=10

Let's make a clear sub-section for Comparison Operators by using this **Telugu_movies table**

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from telugu_movies;
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 301 | Baahubali   | Prabhas           | 2015 |    9.0 |
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
| 303 | Pushpa      | Allu Arjun        | 2021 |    8.2 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
| 305 | Magadheera  | Ram Charan        | 2009 |    8.0 |
+-----+-------------+-------------------+------+--------+
```

1. = Equal to

```
Records: 5  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE year = 2015;
+-----+-----------+---------+------+--------+
| mid | title     | hero    | year | rating |
+-----+-----------+---------+------+--------+
| 301 | Baahubali | Prabhas | 2015 |    9.0 |
+-----+-----------+---------+------+--------+
```

2. > Greater than

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE rating > 8.5;
+-----+-----------+------------------+------+--------+
| mid | title     | hero             | year | rating |
+-----+-----------+------------------+------+--------+
| 301 | Baahubali | Prabhas          | 2015 |    9.0 |
| 302 | RRR       | NTR & Ram Charan | 2022 |    8.7 |
+-----+-----------+------------------+------+--------+
```

3. < Less than

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE year < 2015;
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
```

4)>= Greater than or equal to

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE rating >= 8.5;
+-----+------------+------------------+------+--------+
| mid | title      | hero             | year | rating |
+-----+------------+------------------+------+--------+
| 301 | Baahubali  | Prabhas          | 2015 |    9.0 |
| 302 | RRR        | NTR & Ram Charan | 2022 |    8.7 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
+-----+------------+------------------+------+--------+
3 rows in set (0.0007 sec)
```

5.<= Less than or equal to

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE year <= 2015;
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 301 | Baahubali  | Prabhas    | 2015 |    9.0 |
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
2 rows in set (0.0007 sec)
```

6. != Not equal to

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE hero != 'Prabhas';
+-----+-------------+--------------------+------+--------+
| mid | title       | hero               | year | rating |
+-----+-------------+--------------------+------+--------+
| 302 | RRR         | NTR & Ram Charan   | 2022 |    8.7 |
| 303 | Pushpa      | Allu Arjun         | 2021 |    8.2 |
| 304 | Arjun Reddy | Vijay Devarakonda  | 2017 |    8.5 |
| 305 | Magadheera  | Ram Charan         | 2009 |    8.0 |
+-----+-------------+--------------------+------+--------+
```

7.<> Not equal to (same as !=)

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE title <> 'Pushpa';
+-----+-------------+--------------------+------+--------+
| mid | title       | hero               | year | rating |
+-----+-------------+--------------------+------+--------+
| 301 | Baahubali   | Prabhas            | 2015 |    9.0 |
| 302 | RRR         | NTR & Ram Charan   | 2022 |    8.7 |
| 304 | Arjun Reddy | Vijay Devarakonda  | 2017 |    8.5 |
| 305 | Magadheera  | Ram Charan         | 2009 |    8.0 |
+-----+-------------+--------------------+------+--------+
```

Using AND : This will give the output when all the conditions become true.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> WHERE hero = 'Ram Charan' AND year < 2015;
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
```


2. Using OR

Returns rows if *any one* condition is true.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> WHERE hero = 'Prabhas' OR hero = 'Allu Arjun';
+-----+-----------+------------+------+--------+
| mid | title     | hero       | year | rating |
+-----+-----------+------------+------+--------+
| 301 | Baahubali | Prabhas    | 2015 |    9.0 |
| 303 | Pushpa    | Allu Arjun | 2021 |    8.2 |
+-----+-----------+------------+------+--------+
```

3. Using BETWEEN

👉 **Selects rows between two values (inclusive).**

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> WHERE year BETWEEN 2010 AND 2020;
+-----+------------+------------------+------+--------+
| mid | title      | hero             | year | rating |
+-----+------------+------------------+------+--------+
| 301 | Baahubali  | Prabhas          | 2015 |    9.0 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
+-----+------------+------------------+------+--------+
2 rows in set (0.0007 sec)
```

4. Using NOT BETWEEN

Selects rows outside the range.

```
2 rows in set (0.0007 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> WHERE year NOT BETWEEN 2010 AND 2020;
+-----+------------+------------------+------+--------+
| mid | title      | hero             | year | rating |
+-----+------------+------------------+------+--------+
| 302 | RRR        | NTR & Ram Charan | 2022 |    8.7 |
| 303 | Pushpa     | Allu Arjun       | 2021 |    8.2 |
| 305 | Magadheera | Ram Charan       | 2009 |    8.0 |
+-----+------------+------------------+------+--------+
3 rows in set (0.0021 sec)
```


5. Using IN

👉 **Selects rows matching any value in a list.**

```
3 rows in set (0.0021 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> WHERE hero IN ('Prabhas', 'Ram Charan');
+-----+------------+-----------+------+--------+
| mid | title      | hero      | year | rating |
+-----+------------+-----------+------+--------+
| 301 | Baahubali  | Prabhas   | 2015 |    9.0 |
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+-----------+------+--------+
2 rows in set (0.0013 sec)
```

6. Using NOT IN

👉 Excludes rows with those values.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT * FROM telugu_movies
                                        -> WHERE hero NOT IN ('Prabhas', 'Ram Charan');
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
| 303 | Pushpa      | Allu Arjun        | 2021 |    8.2 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
+-----+-------------+-------------------+------+--------+
```

7. Using IS NULL

👉 Finds rows with NULL values.

(Example: If some rating is missing).

```
3 rows in set (0.0005 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT * FROM telugu_movies
                                        -> WHERE rating IS NULL;
Empty set (0.0021 sec)
MySQL localhost:33060+ ssl 10kcoders SQL >
```

Using IS NOT NULL

👉 Finds rows where column has value.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT * FROM telugu_movies
                                        -> WHERE rating IS NOT NULL;
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 301 | Baahubali   | Prabhas           | 2015 |    9.0 |
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
| 303 | Pushpa      | Allu Arjun        | 2021 |    8.2 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
| 305 | Magadheera  | Ram Charan        | 2009 |    8.0 |
+-----+-------------+-------------------+------+--------+
5 rows in set (0.0004 sec)
MySQL localhost:33060+ ssl 10kcoders SQL >
```

(All 5 movies, since none are NULL)

9. Using LIKE

Used for pattern matching with wildcards:

1. **% → any number of characters**
2. **_ → one character**

Example 1: Movies starting with 'B'

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> WHERE title LIKE 'B%';
+-----+-----------+---------+------+--------+
| mid | title     | hero    | year | rating |
+-----+-----------+---------+------+--------+
| 301 | Baahubali | Prabhas | 2015 |    9.0 |
+-----+-----------+---------+------+--------+
```

Example 2: Movies ending with 'a'

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> WHERE title LIKE '%a';
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 303 | Pushpa     | Allu Arjun | 2021 |    8.2 |
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
2 rows in set (0.0006 sec)
```

Example 3: Movies where 2nd letter is 'a'

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> WHERE title LIKE '_a%';
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 301 | Baahubali  | Prabhas    | 2015 |    9.0 |
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
```

Example 4: Movies containing 'R' in the middle

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> WHERE title LIKE '%R%';
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
| 305 | Magadheera  | Ram Charan        | 2009 |    8.0 |
+-----+-------------+-------------------+------+--------+
3 rows in set (0.0007 sec)
```

Example 5: movies with year 2022

```
1 row in set (0.0011 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies WHERE year LIKE 2022
                                          -> ;
+-----+-------+-----------------+------+--------+
| mid | title | hero            | year | rating |
+-----+-------+-----------------+------+--------+
| 302 | RRR   | NTR & Ram Charan | 2022 |    8.7 |
+-----+-------+-----------------+------+--------+
1 row in set (0.0216 sec)
```

Example 6: This will give the rows whose name's third letter start with 'j'.

```
Empty set (0.0007 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from telugu_movies where title like '__j%';
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
+-----+-------------+-------------------+------+--------+
1 row in set (0.0021 sec)
```

Example 7: This will give the rows whose name contains 2 a's.

```
1 row in set (0.0021 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from telugu_movies
                                          -> where title like '%a%a%';
+-----+------------+------------+------+--------+
| mid | title      | hero       | year | rating |
+-----+------------+------------+------+--------+
| 301 | Baahubali  | Prabhas    | 2015 |    9.0 |
| 305 | Magadheera | Ram Charan | 2009 |    8.0 |
+-----+------------+------------+------+--------+
2 rows in set (0.0012 sec)
```

USING ORDER BY :

This will be used to ordering the columns data (ascending or descending).

Syntax

SELECT * FROM ORDER BY <col _name> DESC;

1. By default MySQL will use ascending order.
2. If you want output in descending order you have to use desc keyword after the column.
3. By default → ASC (ascending order).
4. To reverse → DESC (descending order).

1. **Sort movies by rating (highest first)**

```
2 rows in set (0.0012 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> ORDER BY rating DESC;
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 301 | Baahubali   | Prabhas           | 2015 |    9.0 |
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
| 303 | Pushpa      | Allu Arjun        | 2021 |    8.2 |
| 305 | Magadheera  | Ram Charan        | 2009 |    8.0 |
+-----+-------------+-------------------+------+--------+
```


2. **Sort movies by year (oldest first)**

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                           -> ORDER BY year ASC;
+-----+-------------+-------------------+------+--------+
| mid | title       | hero              | year | rating |
+-----+-------------+-------------------+------+--------+
| 305 | Magadheera  | Ram Charan        | 2009 |    8.0 |
| 301 | Baahubali   | Prabhas           | 2015 |    9.0 |
| 304 | Arjun Reddy | Vijay Devarakonda | 2017 |    8.5 |
| 303 | Pushpa      | Allu Arjun        | 2021 |    8.2 |
| 302 | RRR         | NTR & Ram Charan  | 2022 |    8.7 |
+-----+-------------+-------------------+------+--------+
5 rows in set (0.0010 sec)
```

3. **Sort movies by hero name (alphabetical order)**

```
5 rows in set (0.0010 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> ORDER BY hero ASC;
+-----+-------------+--------------------+------+--------+
| mid | title       | hero               | year | rating |
+-----+-------------+--------------------+------+--------+
| 303 | Pushpa      | Allu Arjun         | 2021 |    8.2 |
| 302 | RRR         | NTR & Ram Charan   | 2022 |    8.7 |
| 301 | Baahubali   | Prabhas            | 2015 |    9.0 |
| 305 | Magadheera  | Ram Charan         | 2009 |    8.0 |
| 304 | Arjun Reddy | Vijay Devarakonda  | 2017 |    8.5 |
+-----+-------------+--------------------+------+--------+
5 rows in set (0.0012 sec)
```

4. **Sort movies by year (latest first) and if same year, by rating**

```
5 rows in set (0.0012 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT * FROM telugu_movies
                                          -> ORDER BY year DESC, rating DESC;
+-----+-------------+--------------------+------+--------+
| mid | title       | hero               | year | rating |
+-----+-------------+--------------------+------+--------+
| 302 | RRR         | NTR & Ram Charan   | 2022 |    8.7 |
| 303 | Pushpa      | Allu Arjun         | 2021 |    8.2 |
| 304 | Arjun Reddy | Vijay Devarakonda  | 2017 |    8.5 |
| 301 | Baahubali   | Prabhas            | 2015 |    9.0 |
| 305 | Magadheera  | Ram Charan         | 2009 |    8.0 |
+-----+-------------+--------------------+------+--------+
5 rows in set (0.0030 sec)
```


Taking Backup of Table Data

Create a Backup Table Using CREATE TABLE AS SELECT

You can create a new table using the data from an existing table. This method creates a copy of the table along with its data.

Syntax:

CREATE TABLE <Backup_Table_Name> AS

SELECT * FROM <Original_Table_Name>;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > Create table emp_bkp as
                                           -> select * from emp;

Query OK, 6 rows affected (0.0844 sec)

Records: 6  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp_bkp;
+-------+-------+------------+----------+
| empid | ename | DOB        | salary   |
+-------+-------+------------+----------+
|   101 | RAM   | 2021-02-22 | 20000.32 |
|   102 | Priya | 2034-05-29 | 25000.25 |
|   103 | Amit  | 2023-03-22 | 40000.32 |
|   104 | Sita  | 2022-05-10 | 25000.50 |
|   105 | Ravi  | 2020-11-15 | 30000.00 |
|   106 | Kiran | 2021-08-01 | 28000.75 |
+-------+-------+------------+----------+
6 rows in set (0.0023 sec)
```

Ex2::

Create Table with Your Own Column Names (Two Steps)

Since MySQL doesn't allow column renaming directly in CREATE TABLE AS SELECT, you need to do it like this:

Step 1: Create empty table with your column names

Step 2: Insert data into it

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > CREATE TABLE emp1_bkp (
                                           ->     empid INT,
                                           ->     ename VARCHAR(50),
                                           ->     salary DECIMAL(10,2)
                                           -> );
Query OK, 0 rows affected (0.0588 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > INSERT INTO emp1_bkp (empid, ename, salary)
                                           -> SELECT empid, ename, salary
                                           -> FROM emp;

Query OK, 6 rows affected (0.0372 sec)

Records: 6  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp1_bkp;
+-------+-------+----------+
| empid | ename | salary   |
+-------+-------+----------+
|   101 | RAM   | 20000.32 |
|   102 | Priya | 25000.25 |
|   103 | Amit  | 40000.32 |
|   104 | Sita  | 25000.50 |
|   105 | Ravi  | 30000.00 |
|   106 | Kiran | 28000.75 |
+-------+-------+----------+
6 rows in set (0.0015 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL >
```

Create Table Without Data

```
6 rows in set (0.0015 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > CREATE TABLE emp_empty AS
                                          -> SELECT * FROM emp
                                          -> WHERE 1 = 2;
Query OK, 0 rows affected (0.0330 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  10kcoders  SQL > desc emp_empty;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| empid  | int           | YES  |     | NULL    |       |
| ename  | varchar(50)   | YES  |     | NULL    |       |
| DOB    | date          | YES  |     | NULL    |       |
| salary | decimal(10,2) | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
4 rows in set (0.0289 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL >
```

1. This creates the structure only, no rows.
2. You cannot specify new column names here in MySQL either.

ALIASES

A.Column Aliases

Syntax: SELECT <original_col> [AS] <Aliases_name> FROM ;

Table Aliases If you are using table aliases, you can use DOT method to the columns.

```
ERROR: 1054: Unknown column 'EmpName' in 'field list'
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT EmpID AS ID, EName AS Name Salary AS Sal FROM EMP;
+-----+-------+----------+
| ID  | Name  | Sal      |
+-----+-------+----------+
| 101 | RAM   | 20000.32 |
| 102 | Priya | 25000.25 |
| 103 | Amit  | 40000.32 |
| 104 | Sita  | 25000.50 |
| 105 | Ravi  | 30000.00 |
| 106 | Kiran | 28000.75 |
+-----+-------+----------+
6 rows in set (0.0018 sec)
```

B. Table Aliases

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT E.EmpID, E.EName, E.Salary
                                          -> FROM EMP E;
+-------+-------+----------+
| EmpID | EName | Salary   |
+-------+-------+----------+
|   101 | RAM   | 20000.32 |
|   102 | Priya | 25000.25 |
|   103 | Amit  | 40000.32 |
|   104 | Sita  | 25000.50 |
|   105 | Ravi  | 30000.00 |
|   106 | Kiran | 28000.75 |
+-------+-------+----------+
6 rows in set (0.0217 sec)
```

SQL FUNCTIONS

SQL functions are predefined operations provided by the database. They allow you to perform calculations, manipulate data, and retrieve results in SQL queries.

Functions are broadly classified into three categories:

1 Row-Level Functions (Single-Row Functions)

A. Operate on each row individually.

B. Return one result per row.

C. Common types: Character, Numeric, Date, Conversion.

2 Group Functions (Aggregate Functions)

A. Operate **on multiple rows**.

B. Return **one result for a group of rows**.

C. Often used with the GROUP BY clause.

3 Analytical Functions

A. Perform calculations across **a set of rows** related to the **current row**.

B. Do **not reduce rows** like aggregate functions; return a value **for each row**.

C. Useful for **rankings, running totals, and moving averages**.

1)Row-Level Functions (Single-Row Functions)

1. **Character functions** → manipulate text
2. **Numeric functions** → operate on numbers
3. **Date functions** → operate on dates
4. **Conversion functions** → change data type

MySQL Character (String) Functions

MySQL provides many functions to manipulate text/strings.

1 UPPER() / UCASE()

- Converts all letters to uppercase.

```
SELECT UPPER(ename) AS UpperName FROM emp;
-- OR
SELECT UCASE(ename) AS UpperName FROM emp;
```

```
MySQL  localhost:33060+ ssl  10kcoders  SQL  SELECT UPPER(ename) AS UpperName FROM emp;
+-----------+
| UpperName |
+-----------+
| RAM       |
| PRIYA     |
| AMIT      |
| SITA      |
| RAVI      |
| KIRAN     |
+-----------+
```


2 LOWER() / LCASE()

- Converts all letters to lowercase.

```
SELECT LOWER(ename) AS LowerName FROM emp;
-- OR
SELECT LCASE(ename) AS LowerName FROM emp;
```

3 CONCAT()

- Concatenates two or more strings.

```
SELECT CONCAT(ename, ' works in dept ', deptno) AS Info FROM emp;
```

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT CONCAT(ename, ' employee_salary is ', salary) AS Info FROM emp;
+---------------------------------+
| Info                            |
+---------------------------------+
| RAM employee_salary is 20000.32 |
| Priya employee_salary is 25000.25 |
| Amit employee_salary is 40000.32 |
| Sita employee_salary is 25000.50 |
| Ravi employee_salary is 30000.00 |
| Kiran employee_salary is 28000.75 |
+---------------------------------+
6 rows in set (0.0019 sec)
```

4 CONCAT_WS()

A. Concatenates strings with a separator

B. WS = With Separator

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT CONCAT_WS('-', ename, salary) AS Info FROM emp;
+-----------------+
| Info            |
+-----------------+
| RAM-20000.32    |
| Priya-25000.25  |
| Amit-40000.32   |
| Sita-25000.50   |
| Ravi-30000.00   |
| Kiran-28000.75  |
+-----------------+
6 rows in set (0.0021 sec)
```

5 SUBSTRING() / SUBSTR()

A. Extracts a substring from a string.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT SUBSTRING(ename,1,3) AS ShortName FROM emp;
+-----------+
| ShortName |
+-----------+
| RAM       |
| Pri       |
| Ami       |
| Sit       |
| Rav       |
| Kir       |
+-----------+
6 rows in set (0.0010 sec)
```

6 LENGTH()

A. Returns number of bytes in a string.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT LENGTH(ename) AS NameLength FROM emp;
+------------+
| NameLength |
+------------+
|          3 |
|          5 |
|          4 |
|          4 |
|          4 |
|          5 |
+------------+
6 rows in set (0.0014 sec)
```

7 CHAR_LENGTH() / LENGTH()

- Returns number of characters in a string (handles multibyte).

SELECT CHAR_LENGTH(ename) AS NameCharCount FROM emp;

8 TRIM()

- Removes leading and trailing spaces (or specified characters).

```
6 rows in set (0.0050 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT TRIM(ename) AS TrimmedName FROM emp;
+-------------+
| TrimmedName |
+-------------+
| RAM         |
| Priya       |
| Amit        |
| Sita        |
| Ravi        |
| Kiran       |
+-------------+
```

-- Remove specific character

```
Query OK, 0 rows affected (0.0004 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT TRIM('X' FROM 'XXXabcXXX') AS Cleaned;
+---------+
| Cleaned |
+---------+
| abc     |
+---------+
1 row in set (0.0006 sec)
MySQL localhost:33060+ ssl 10kcoders SQL >
```

9 LTRIM() / RTRIM()

- LTRIM() → Removes leading spaces
- RTRIM() → Removes trailing spaces

SELECT LTRIM(ename) AS LeftTrimmed FROM emp;

SELECT RTRIM(ename) AS RightTrimmed FROM emp;

10 REPLACE()

- Replaces substring with another string.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT REPLACE(ename, 'A', '@') AS ReplacedName FROM emp;
+--------------+
| ReplacedName |
+--------------+
| R@M          |
| Priya        |
| @mit         |
| Sita         |
| Ravi         |
| Kiran        |
+--------------+
```

11 INSTR()

- Returns position of substring (1-based).

```
6 rows in set (0.0012 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT INSTR(ename,'A') AS Pos FROM emp;
+-----+
| Pos |
+-----+
|   2 |
|   5 |
|   1 |
|   4 |
|   2 |
|   4 |
+-----+
6 rows in set (0.0212 sec)
```

12 LOCATE()

- Returns position of substring, similar to INSTR().

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT LOCATE('A', ename) AS Pos FROM emp;
+-----+
| Pos |
+-----+
|   2 |
|   5 |
|   1 |
|   4 |
|   2 |
|   4 |
+-----+
```

13 LEFT() / RIGHT()

- Extract leftmost or rightmost characters.

```
6 rows in set (0.0013 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT LEFT(ename,3) AS LeftPart FROM emp;
+----------+
| LeftPart |
+----------+
| RAM      |
| Pri      |
| Ami      |
| Sit      |
| Rav      |
| Kir      |
+----------+
6 rows in set (0.0040 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT RIGHT(ename,3) AS RightPart FROM emp;
+-----------+
| RightPart |
+-----------+
| RAM       |
| iya       |
| mit       |
| ita       |
| avi       |
| ran       |
+-----------+
6 rows in set (0.0014 sec)
```

14 REVERSE()

- Reverses the string.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT REVERSE(ename) AS ReversedName FROM emp;
+--------------+
| ReversedName |
+--------------+
| MAR          |
| ayirP        |
| timA         |
| atiS         |
| ivaR         |
| nariK        |
+--------------+
```

15 RPAD() / LPAD()

- Pads a string to desired length with a character.

```
6 rows in set (0.0018 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT LPAD(ename, 10, '*') AS PaddedLeft FROM emp;
+------------+
| PaddedLeft |
+------------+
| *******RAM |
| *****Priya |
| ******Amit |
| ******Sita |
| ******Ravi |
| *****Kiran |
+------------+
6 rows in set (0.0040 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT RPAD(ename, 10, '*') AS PaddedRight FROM emp;
+-------------+
| PaddedRight |
+-------------+
| RAM******* |
| Priya***** |
| Amit****** |
| Sita****** |
| Ravi****** |
| Kiran***** |
+-------------+
6 rows in set (0.0009 sec)
```

16 FIELD()

- Returns position of string in a list of strings.

```
6 rows in set (0.0220 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT FIELD(ename,'ram','priya','ravi') AS Position FROM emp;
+----------+
| Position |
+----------+
|        1 |
|        2 |
|        0 |
|        0 |
|        3 |
|        0 |
+----------+
6 rows in set (0.0010 sec)
```


17 FORMAT() (for numbers, optional with strings)

- Formats numbers as strings with commas/decimals.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT FORMAT(salary,2) AS FormattedSal FROM emp;
+--------------+
| FormattedSal |
+--------------+
| 20,000.32    |
| 25,000.25    |
| 40,000.32    |
| 25,000.50    |
| 30,000.00    |
| 28,000.75    |
+--------------+
6 rows in set (0.0011 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT FORMAT(salary,3) AS FormattedSal FROM emp;
+--------------+
| FormattedSal |
+--------------+
| 20,000.320   |
| 25,000.250   |
| 40,000.320   |
| 25,000.500   |
| 30,000.000   |
| 28,000.750   |
+--------------+
6 rows in set (0.0014 sec)
```

It'll be one query showing all functions on your EMP table.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT UPPER(ename), CONCAT(ename,'-',salary), SUBSTRING(ename,1,3), REPLACE(ename,'A','@') FROM emp;
+-------------+-------------------------+--------------------+----------------------+
| UPPER(ename) | CONCAT(ename,'-',salary) | SUBSTRING(ename,1,3) | REPLACE(ename,'A','@') |
+-------------+-------------------------+--------------------+----------------------+
| RAM         | RAM-20000.32            | RAM                | R@M                  |
| PRIYA       | Priya-25000.25          | Pri                | Priya                |
| AMIT        | Amit-40000.32           | Ami                | @mit                 |
| SITA        | Sita-25000.50           | Sit                | Sita                 |
| RAVI        | Ravi-30000.00           | Rav                | Ravi                 |
| KIRAN       | Kiran-28000.75          | Kir                | Kiran                |
+-------------+-------------------------+--------------------+----------------------+
6 rows in set (0.0029 sec)
```

18 INSERT(s,pos,len,new)

- Replaces len characters starting at pos with new string.

```
6 rows in set (0.0029 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT INSERT('hello',2,3,'y') AS Result;
+--------+
| Result |
+--------+
| hyo    |
+--------+
1 row in set (0.0213 sec)
```

19.REPEAT(s,n)

- **Repeats the string n times.**

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT REPEAT('Hi',3) AS Result;
+--------+
| Result |
+--------+
| HiHiHi |
+--------+
1 row in set (0.0017 sec)
```

20. SPACE(n)

- **Returns n spaces.**

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT CONCAT('A',SPACE(5),'B') AS Result;
+---------+
| Result  |
+---------+
| A     B |
+---------+
1 row in set (0.0211 sec)
```

MySQL Character (String) Functions Table

Function	**Description**	**Example**	**Result**
	s ASCII value of first character	ASCII('A')	65
CHAR_LENGTH(s)	Number of characters (not bytes)	CHAR_LENGTH('MySQL')	5
LENGTH(s)	String length in bytes	LENGTH('MySQL')	5
CONCAT(s1,s2,...)	Joins strings together	CONCAT('My','SQL')	MySQL
CONCAT_WS(sep,s1,s2,...)	Joins strings with separator	CONCAT_WS('-', '2025','08','14')	2025-08-14
FORMAT(n,d)	Formats number with commas & decimals	FORMAT(12345.678,2)	12,345.68
INSERT(s,pos,len,newstr)	Replace len characters starting at pos with newstr	INSERT('hello',2,3,'y')	hyo
INSTR(s,substr)	Returns position of substring	INSTR('hello','l')	3
LOCATE(substr,s)	Returns position of substring	LOCATE('SQL','MySQL')	3
LEFT(s,n)	Leftmost n characters	LEFT('MySQL',2)	My
RIGHT(s,n)	Rightmost n characters	RIGHT('MySQL',2)	QL
LOWER(s) / LCASE(s)	Converts string to lowercase	LOWER('MySQL')	mysql
UPPER(s) / UCASE(s)	Converts string to uppercase	UPPER('mysql')	MYSQL
LPAD(s,len,pad)	Pads string on the left to total length len	LPAD('SQL',5,'*')	**SQL
RPAD(s,len,pad)	Pads string on the right to total length len	RPAD('SQL',5,'*')	SQL**
LTRIM(s)	Remove spaces from left	LTRIM(' SQL')	SQL
RTRIM(s)	Remove spaces from right	RTRIM('SQL ')	SQL
TRIM(s)	Remove spaces from both sides	TRIM(' SQL ')	SQL
POSITION(substr IN s)	Returns position of substring	POSITION('SQL' IN 'MySQL')	3
REPEAT(s,n)	Repeat string n times	REPEAT('Hi',3)	HiHiHi
REPLACE(s,old,new)	Replace substring	REPLACE('Hello','l','p')	Heppo

REVERSE(s)	Reverse the string	REVERSE('MySQL')	LQSyM
SPACE(n)	Returns a string with n spaces	CONCAT('A',SPACE(5),'B')	A B
SUBSTRING(s,pos,len) / SUBSTR(s,pos,len)	Extract substring starting at pos with length len	SUBSTRING('MySQL',2,3)	ySQ

MySQL Date & Time Functions

Function	Example	Result
CURDATE()	SELECT CURDATE();	2025-08-14
CURRENT_DATE()	SELECT CURRENT_DATE();	2025-08-14
CURTIME()	SELECT CURTIME();	18:45:23
CURRENT_TIME()	SELECT CURRENT_TIME();	18:45:23
NOW()	SELECT NOW();	2025-08-14 18:45:23
SYSDATE()	SELECT SYSDATE();	2025-08-14 18:45:23
DATE(expr)	SELECT DATE('2025-08-14 10:30:00');	2025-08-14
TIME(expr)	SELECT TIME('2025-08-14 10:30:00');	10:30:00
YEAR(date)	SELECT YEAR('2025-08-14');	2025
MONTH(date)	SELECT MONTH('2025-08-14');	8
DAY(date)	SELECT DAY('2025-08-14');	14
DAYNAME(date)	SELECT DAYNAME('2025-08-14');	Thursday
DAYOFWEEK(date)	SELECT DAYOFWEEK('2025-08-14');	5
DAYOFYEAR(date)	SELECT DAYOFYEAR('2025-08-14');	226
HOUR(time)	SELECT HOUR('18:45:23');	18
MINUTE(time)	SELECT MINUTE('18:45:23');	45
SECOND(time)	SELECT SECOND('18:45:23');	23
ADDDATE(date, INTERVAL n unit)	SELECT ADDDATE('2025-08-14', INTERVAL 5 DAY);	2025-08-19
SUBDATE(date, INTERVAL n unit)	SELECT SUBDATE('2025-08-14', INTERVAL 5 DAY);	2025-08-09

DATEDIFF(date1, date2)	SELECT DATEDIFF('2025-08-14','2025-08-10');	4
Function	**Example**	**Result**
TIMEDIFF(time1, time2)	SELECT TIMEDIFF('18:00:00','15:30:00');	02:30:00
LAST_DAY(date)	SELECT LAST_DAY('2025-08-14');	2025-08-31
MAKEDATE(year, day_of_year)	SELECT MAKEDATE(2025, 32);	2025-02-01
QUARTER(date)	SELECT QUARTER('2025-08-14');	3
WEEK(date)	SELECT WEEK('2025-08-14');	32
MONTHNAME(date)	SELECT MONTHNAME('2025-08-14');	August

SQL Conversion Functions

Function	**Description**	**Example**	**Result**
`TO_CHAR(number	date, format)`	Convert number or date to string in specified format	SELECT TO_CHAR(SYSDATE,'DD-MON-YYYY');
TO_NUMBER(string)	Convert string to number	SELECT TO_NUMBER('1234');	1234
TO_DATE(string, format)	Convert string to date using format	SELECT TO_DATE('04-10-2025','DD-MM-YYYY');	2025-10-04
CAST(expr AS type)	Convert expression to another datatype	SELECT CAST(1500 AS CHAR(10));	'1500'

Numeric/Mathematical Functions

Function	Description	Example
ABS(number)	Absolute value	SELECT ABS(-10) FROM dual; → 10
CEIL(number)	Smallest integer ≥ number	SELECT CEIL(2.3) FROM dual; → 3
FLOOR(number)	Largest integer ≤ number	SELECT FLOOR(2.7) FROM dual; → 2
ROUND(number, decimal_places)	Round number	SELECT ROUND(12.345,2) FROM dual; → 12.35
TRUNC(number, decimal_places)	Truncate number	SELECT TRUNC(12.345,2) FROM dual; → 12.34
MOD(number1, number2)	Modulus	SELECT MOD(10,3) FROM dual; → 1
POWER(number, power)	Raise to power	SELECT POWER(2,3) FROM dual; → 8
SQRT(number)	Square root	SELECT SQRT(16) FROM dual; → 4
EXP(number)	e^number	SELECT EXP(1) FROM dual; → 2.71828
LN(number)	Natural log	SELECT LN(10) FROM dual;
LOG(number)	Log base 10	SELECT LOG(100) FROM dual; → 2

Aggregate functions in MySQL perform calculations on multiple rows and return a single summarized value.

The commonly used aggregate functions include:

- **SUM**(column_name): Calculates the total sum of values in a numeric column.
- **MAX**(column_name): Returns the maximum value in a column.
- **MIN**(column_name): Returns the minimum value in a column.
- **AVG**(column_name): Calculates the average (mean) value of a numeric column.
- **COUNT**(column_name or *): Counts the number of rows, or distinct values matching a condition.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp;
+-------+----------+------------------+------+------------+------+------+--------+
| empno | ename    | job              | mgr  | hiredate   | sal  | comm | deptno |
+-------+----------+------------------+------+------------+------+------+--------+
|  3717 | Arjun    | python_developer | NULL | 1999-05-29 | 4000 | NULL |   NULL |
|  7369 | AMMU     | python_developer | 7902 | 1980-12-17 |  800 |    0 |     20 |
|  7499 | Amit     | Technical Trainer| 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | Rajesh   | Technical Trainer| 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7566 | Bujji    | MANAGER          | 7839 | 1981-04-02 | 2975 |    0 |     20 |
|  7654 | chaitanya| Technical Trainer| 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7698 | Suresh   | MANAGER          | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7782 | Krishna  | MANAGER          | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7788 | Anil     | ANALYST          | 7566 | 1982-12-09 | 3000 |    0 |     20 |
|  7839 | Rakesh   | PRESIDENT        | NULL | 1981-11-17 | 5000 |    0 |     10 |
|  7844 | Rohit    | Technical Trainer| 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7876 | Kiran    | python_developer | 7788 | 1983-05-23 | 1100 |    0 |     20 |
|  7900 | Aparna   | python_developer | 7698 | 1981-12-03 | 3000 |    0 |     30 |
|  7902 | Prachi   | ANALYST          | 7566 | 1981-12-03 | 3000 |    0 |     20 |
|  7934 | Sunita   | python_developer | 7782 | 1982-01-23 | 1300 |    0 |     10 |
+-------+----------+------------------+------+------------+------+------+--------+
```

-- NOTE: From now onwards, we will use this EMP table as our standard table for practice.

-- Owner: 10Kcoders Training

```
ALTER TABLE EMP COMMENT = 'This EMP table will be used from now onwards (owner: 10Kcoders)';
```

Aggregate functions.

1. SUM()

- **Purpose:** Calculates the total of a numeric column.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT SUM(sal) AS Total_Salary FROM emp;
+--------------+
| Total_Salary |
+--------------+
|        35075 |
+--------------+
1 row in set (0.0224 sec)
```

Returns the **sum of all salaries**.

2. MAX() – Highest Salary

SELECT MAX(sal) AS Highest_Salary

FROM emp;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT MAX(sal) AS Highest_Salary
                                           -> FROM emp;

+----------------+
| Highest_Salary |
+----------------+
|           5000 |
+----------------+
```

Returns **5000** (Rakesh's salary).

3. MIN() – Lowest Salary

SELECT MIN(sal) AS Lowest_Salary

FROM emp;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT MIN(sal) AS Lowest_Salary
                                           -> FROM emp;

+---------------+
| Lowest_Salary |
+---------------+
|           800 |
+---------------+
1 row in set (0.0028 sec)
```

Returns **800**(Ammu's salary).

4.AVG() – Average Salary

```
1 row in set (0.0028 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT AVG(sal) AS Average_Salary
                                           -> FROM emp;

+----------------+
| Average_Salary |
+----------------+
|      2338.3333 |
+----------------+
1 row in set (0.0026 sec)
```

Returns the average salary of all employees.

5. COUNT() – Number of Employees

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT COUNT(*) AS Total_Employees
                                           -> FROM emp;

+-----------------+
| Total_Employees |
+-----------------+
|              15 |
+-----------------+
```

Returns 15 (total rows).

Group functions in MySQL—also known as aggregate functions—operate on multiple rows to return a single summarized value, typically used with the GROUP BY clause to aggregate data for each group. The most commonly used group (aggregate) functions in MySQL are listed below.Using GROUP BY (Per Department, Per Job)

Average Salary per Department

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT deptno, AVG(sal) AS Avg_Salary
                                           -> FROM emp
                                           -> GROUP BY deptno;

+--------+------------+
| deptno | Avg_Salary |
+--------+------------+
|   NULL |  4000.0000 |
|     20 |  2175.0000 |
|     30 |  1908.3333 |
|     10 |  2916.6667 |
+--------+------------+
```

Returns average salary for each department (10, 20, 30).

Maximum Salary per Job

```
GROUP BY job' at line 1
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT job, MAX(sal) AS Max_Salary
                                           -> FROM emp
                                           -> GROUP BY job;

+-------------------+------------+
| job               | Max_Salary |
+-------------------+------------+
| python_developer  |       4000 |
| Technical Trainer |       1600 |
| MANAGER           |       2975 |
| ANALYST           |       3000 |
| PRESIDENT         |       5000 |
+-------------------+------------+
```

Shows the highest salary in each job role.

Count of Employees per Department

```
5 rows in set (0.0018 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT deptno, COUNT(*) AS Total_Employees
                                           -> FROM emp
                                           -> GROUP BY deptno;

+--------+-----------------+
| deptno | Total_Employees |
+--------+-----------------+
|   NULL |               1 |
|     20 |               5 |
|     30 |               6 |
|     10 |               3 |
+--------+-----------------+
```

Returns how many employees are in each department.

Sum of Salaries per Department

```
4 rows in set (0.0021 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT deptno, SUM(sal) AS Total_Salary
                                          -> FROM emp
                                          -> GROUP BY deptno;

+--------+--------------+
| deptno | Total_Salary |
+--------+--------------+
|   NULL |         4000 |
|     20 |        10875 |
|     30 |        11450 |
|     10 |         8750 |
+--------+--------------+
4 rows in set (0.0017 sec)
```

Returns total salary per department**.**

Example 2 – Average salary per job

```
15 rows in set (0.0222 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT job, AVG(sal) AS Avg_Salary
                                          -> FROM emp
                                          -> GROUP BY job
                                          -> ORDER BY job;

+-------------------+------------+
| job               | Avg_Salary |
+-------------------+------------+
| ANALYST           |  2822.5000 |
| MANAGER           |  2758.3333 |
| PRESIDENT         |  5000.0000 |
| python_developer  |  2040.0000 |
| Technical Trainer |  1400.0000 |
+-------------------+------------+
```

2. HAVING Clause

- **Definition:**

  HAVING is like a **WHERE for groups**.

  It is used **after GROUP BY** to filter groups, since aggregate functions cannot be used in WHERE.

Having

• This will work as where clause which can be used only with group by because of absence of where clause in group by.

• On top of the Group by if we want to filter the groups then we use having clause. Example: Query to find the duplicate records with count.

Syntax:

SELECT column_name, AGG_FUNCTION(column_name)

FROM table_name

GROUP BY column_name

HAVING condition;

Example 1 – Departments with average salary > 2500

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT deptno, AVG(sal) AS Avg_Salary
                                           -> FROM emp
                                           -> GROUP BY deptno
                                           -> HAVING AVG(sal) > 2500;
+--------+------------+
| deptno | Avg_Salary |
+--------+------------+
|   NULL |  4000.0000 |
|     10 |  2916.6667 |
+--------+------------+
```

Example 2– Find duplicate employees (if any)

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT job, COUNT(*) AS Job_Count
                                           -> FROM emp
                                           -> GROUP BY job
                                           -> HAVING COUNT(*) > 1;
+-------------------+-----------+
| job               | Job_Count |
+-------------------+-----------+
| python_developer  |         5 |
| Technical Trainer |         4 |
| MANAGER           |         3 |
| ANALYST           |         2 |
+-------------------+-----------+
```

Order of Execution in GROUP BY + HAVING

1. **GROUP BY** → Rows are grouped.
2. **Aggregate Functions** → Calculated for each group.
3. **HAVING** → Filters the groups.
4. **ORDER BY** → Sorts the final result.

Question 1: Display the Top 3 Highest Paid Employees

Goal: Use ORDER BY and LIMIT to get top salaries.

Query:

```
4 rows in set (0.0022 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, job, sal
                                           -> FROM emp
                                           -> ORDER BY sal DESC
                                           -> LIMIT 3;

+--------+------------------+------+
| ename  | job              | sal  |
+--------+------------------+------+
| Rakesh | PRESIDENT        | 5000 |
| Arjun  | python_developer | 4000 |
| Aparna | python_developer | 3000 |
+--------+------------------+------+
```

Explanation:

- ORDER BY sal DESC → sorts salaries from highest to lowest.
- LIMIT 3 → shows only the **top 3** rows.

Question 2: Display the 4th and 5th Highest Paid Employees

Goal: Use `LIMIT` with `OFFSET` to skip rows.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, job, sal
                                           -> FROM emp
                                           -> ORDER BY sal DESC
                                           -> LIMIT 2 OFFSET 3;

+--------+---------+------+
| ename  | job     | sal  |
+--------+---------+------+
| Prachi | ANALYST | 3000 |
| Bujji  | MANAGER | 2975 |
+--------+---------+------+
2 rows in set (0.0075 sec)
```

Explanation:

1. ORDER BY sal DESC → sorts by highest salary first.
2. OFFSET 3 → skips first 3 rows (top 3 highest salaries).
3. LIMIT 2 → then shows next 2 rows.

Analytical (Window) Functions in MySQL

1. What are Analytical (Window) Functions?

- They perform calculations **across a set of rows related to the current row** (a "window").
- Unlike aggregate functions, **they don't collapse rows into one** — instead, they return a value **for every row**.

Syntax

<function_name>() OVER (

[PARTITION BY column_name]

[ORDER BY column_name]

)

PARTITION BY → divides the result set into groups (like GROUP BY, but doesn't collapse).

ORDER BY → defines the order within each partition (important for ranking and running totals).

Function	PARTITION BY	ORDER BY	Example Usage
SUM()	Optional	Optional	Running totals
RANK()/Dense_Rank()	Optional	☑ Required	Salary rank
ROW_NUMBER()	Optional	☑ Required	Unique row ID
LAG/LEAD	Optional	☑ Required	Prev/Next row values
NTILE()	Optional	☑ Required	Divide into groups

ROW_NUMBER()

Assigns a unique sequential number to each row.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, job, sal,
                                          ->       ROW_NUMBER() OVER (ORDER BY sal DESC) AS row_num
                                          -> FROM emp;
+-----------+------------------+------+---------+
| ename     | job              | sal  | row_num |
+-----------+------------------+------+---------+
| Rakesh    | PRESIDENT        | 5000 |       1 |
| Arjun     | python_developer | 4000 |       2 |
| Aparna    | python_developer | 3000 |       3 |
| Prachi    | ANALYST          | 3000 |       4 |
| Bujji     | MANAGER          | 2975 |       5 |
| Suresh    | MANAGER          | 2850 |       6 |
| Anil      | ANALYST          | 2645 |       7 |
| Krishna   | MANAGER          | 2450 |       8 |
| Amit      | Technical Trainer| 1600 |       9 |
| Rohit     | Technical Trainer| 1500 |      10 |
| jahnavi   | python_developer | 1300 |      11 |
| Rajesh    | Technical Trainer| 1250 |      12 |
| chaitanya | Technical Trainer| 1250 |      13 |
| Kiran     | python_developer | 1100 |      14 |
| Ammu      | python_developer |  800 |      15 |
+-----------+------------------+------+---------+
15 rows in set (0.0435 sec)
```

Output:

Ranks employees by salary (1 = highest, 2 = next, …).

If two people have same salary, they get different numbers.

RANK()

Gives same rank for same salary but **skips the next number**.

```
15 rows in set (0.0435 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, job, sal,
                                          ->        RANK() OVER (ORDER BY sal DESC) AS rank_no
                                          -> FROM emp;
+-----------+-------------------+------+---------+
| ename     | job               | sal  | rank_no |
+-----------+-------------------+------+---------+
| Rakesh    | PRESIDENT         | 5000 |       1 |
| Arjun     | python_developer  | 4000 |       2 |
| Aparna    | python_developer  | 3000 |       3 |
| Prachi    | ANALYST           | 3000 |       3 |
| Bujji     | MANAGER           | 2975 |       5 |
| Suresh    | MANAGER           | 2850 |       6 |
| Anil      | ANALYST           | 2645 |       7 |
| Krishna   | MANAGER           | 2450 |       8 |
| Amit      | Technical Trainer | 1600 |       9 |
| Rohit     | Technical Trainer | 1500 |      10 |
| jahnavi   | python_developer  | 1300 |      11 |
| Rajesh    | Technical Trainer | 1250 |      12 |
| chaitanya | Technical Trainer | 1250 |      12 |
| Kiran     | python_developer  | 1100 |      14 |
| Ammu      | python_developer  |  800 |      15 |
+-----------+-------------------+------+---------+
15 rows in set (0.0057 sec)
```

Example:

If 2 people have sal = 3000, both get **rank 2**, next one gets **rank 4** (rank 3 skipped).

DENSE_RANK()

Same as RANK(), but **no gaps** in rank numbers.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, job, sal,
                                          ->        DENSE_RANK() OVER (ORDER BY sal DESC) AS dense_rank_no
                                          -> FROM emp;
+-----------+-------------------+------+---------------+
| ename     | job               | sal  | dense_rank_no |
+-----------+-------------------+------+---------------+
| Rakesh    | PRESIDENT         | 5000 |             1 |
| Arjun     | python_developer  | 4000 |             2 |
| Aparna    | python_developer  | 3000 |             3 |
| Prachi    | ANALYST           | 3000 |             3 |
| Bujji     | MANAGER           | 2975 |             4 |
| Suresh    | MANAGER           | 2850 |             5 |
| Anil      | ANALYST           | 2645 |             6 |
| Krishna   | MANAGER           | 2450 |             7 |
| Amit      | Technical Trainer | 1600 |             8 |
| Rohit     | Technical Trainer | 1500 |             9 |
| jahnavi   | python_developer  | 1300 |            10 |
| Rajesh    | Technical Trainer | 1250 |            11 |
| chaitanya | Technical Trainer | 1250 |            11 |
| Kiran     | python_developer  | 1100 |            12 |
| Ammu      | python_developer  |  800 |            13 |
+-----------+-------------------+------+---------------+
```

Example:

If two have same salary = 3000 → both rank 2, next one gets rank 3.

Used to **access next or previous row's value** without self join.

LAG() – Previous row

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal,
                                          ->        LAG(sal) OVER (ORDER BY sal) AS prev_sal
                                          -> FROM emp;
+-----------+------+----------+
| ename     | sal  | prev_sal |
+-----------+------+----------+
| Ammu      |  800 |     NULL |
| Kiran     | 1100 |      800 |
| Rajesh    | 1250 |     1100 |
| chaitanya | 1250 |     1250 |
| jahnavi   | 1300 |     1250 |
| Rohit     | 1500 |     1300 |
| Amit      | 1600 |     1500 |
| Krishna   | 2450 |     1600 |
| Anil      | 2645 |     2450 |
| Suresh    | 2850 |     2645 |
| Bujji     | 2975 |     2850 |
| Aparna    | 3000 |     2975 |
| Prachi    | 3000 |     3000 |
| Arjun     | 4000 |     3000 |
| Rakesh    | 5000 |     4000 |
+-----------+------+----------+
```

Shows each employee's previous salary in order.

LEAD() – Next row

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal,
                                          ->        LEAD(sal) OVER (ORDER BY sal) AS next_sal
                                          -> FROM emp;
+-----------+------+----------+
| ename     | sal  | next_sal |
+-----------+------+----------+
| Ammu      |  800 |     1100 |
| Kiran     | 1100 |     1250 |
| Rajesh    | 1250 |     1250 |
| chaitanya | 1250 |     1300 |
| jahnavi   | 1300 |     1500 |
| Rohit     | 1500 |     1600 |
| Amit      | 1600 |     2450 |
| Krishna   | 2450 |     2645 |
| Anil      | 2645 |     2850 |
| Suresh    | 2850 |     2975 |
| Bujji     | 2975 |     3000 |
| Aparna    | 3000 |     3000 |
| Prachi    | 3000 |     4000 |
| Arjun     | 4000 |     5000 |
| Rakesh    | 5000 |     NULL |
+-----------+------+----------+
```

Shows each employee's next salary in order.

NTILE(n)

Divides rows into **n equal parts (buckets)** and numbers each row with the bucket number.

```
15 rows in set (0.0001 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal,
                                          ->        NTILE(4) OVER (ORDER BY sal DESC) AS quartile
                                          -> FROM emp;
+-----------+------+----------+
| ename     | sal  | quartile |
+-----------+------+----------+
| Rakesh    | 5000 |        1 |
| Arjun     | 4000 |        1 |
| Aparna    | 3000 |        1 |
| Prachi    | 3000 |        1 |
| Bujji     | 2975 |        2 |
| Suresh    | 2850 |        2 |
| Anil      | 2645 |        2 |
| Krishna   | 2450 |        2 |
| Amit      | 1600 |        3 |
| Rohit     | 1500 |        3 |
| jahnavi   | 1300 |        3 |
| Rajesh    | 1250 |        3 |
| chaitanya | 1250 |        4 |
| Kiran     | 1100 |        4 |
| Ammu      |  800 |        4 |
+-----------+------+----------+
```

Example:

Divides employees into **4 salary groups (quartiles)** — top 25%, next 25%, etc.

Aggregate Functions as Window Functions

You can use any aggregate function like SUM(), AVG(), MIN(), MAX(), or COUNT() inside a window.

Running Total (Cumulative SUM)

```
15 rows in set (0.0127 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal,
                                          ->        SUM(sal) OVER (ORDER BY sal) AS running_total
                                          -> FROM emp;
+-----------+------+---------------+
| ename     | sal  | running_total |
+-----------+------+---------------+
| Ammu      |  800 |           800 |
| Kiran     | 1100 |          1900 |
| Rajesh    | 1250 |          4400 |
| chaitanya | 1250 |          4400 |
| jahnavi   | 1300 |          5700 |
| Rohit     | 1500 |          7200 |
| Amit      | 1600 |          8800 |
| Krishna   | 2450 |         11250 |
| Anil      | 2645 |         13895 |
| Suresh    | 2850 |         16745 |
| Bujji     | 2975 |         19720 |
| Aparna    | 3000 |         25720 |
| Prachi    | 3000 |         25720 |
| Arjun     | 4000 |         29720 |
| Rakesh    | 5000 |         34720 |
+-----------+------+---------------+
15 rows in set (0.0100 sec)
```

Adds salary row by row as sorted.

Average Salary per Department

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, deptno, sal,
                                          ->        AVG(sal) OVER (PARTITION BY deptno) AS avg_dept_salary
                                          -> FROM emp;
+-----------+--------+------+-----------------+
| ename     | deptno | sal  | avg_dept_salary |
+-----------+--------+------+-----------------+
| Arjun     |   NULL | 4000 |       4000.0000 |
| Krishna   |     10 | 2450 |       2916.6667 |
| Rakesh    |     10 | 5000 |       2916.6667 |
| jahnavi   |     10 | 1300 |       2916.6667 |
| Ammu      |     20 |  800 |       2104.0000 |
| Bujji     |     20 | 2975 |       2104.0000 |
| Anil      |     20 | 2645 |       2104.0000 |
| Kiran     |     20 | 1100 |       2104.0000 |
| Prachi    |     20 | 3000 |       2104.0000 |
| Amit      |     30 | 1600 |       1908.3333 |
| Rajesh    |     30 | 1250 |       1908.3333 |
| chaitanya |     30 | 1250 |       1908.3333 |
| Suresh    |     30 | 2850 |       1908.3333 |
| Rohit     |     30 | 1500 |       1908.3333 |
| Aparna    |     30 | 3000 |       1908.3333 |
+-----------+--------+------+-----------------+
```

Each employee sees their department's average salary.

Minimum & Maximum per Department

```
15 rows in set (0.0062 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, deptno, sal,
                                          ->        MIN(sal) OVER (PARTITION BY deptno) AS min_dept_sal,
                                          ->        MAX(sal) OVER (PARTITION BY deptno) AS max_dept_sal
                                          -> FROM emp;
+-----------+--------+------+--------------+--------------+
| ename     | deptno | sal  | min_dept_sal | max_dept_sal |
+-----------+--------+------+--------------+--------------+
| Arjun     |   NULL | 4000 |         4000 |         4000 |
| Krishna   |     10 | 2450 |         1300 |         5000 |
| Rakesh    |     10 | 5000 |         1300 |         5000 |
| jahnavi   |     10 | 1300 |         1300 |         5000 |
| Ammu      |     20 |  800 |          800 |         3000 |
| Bujji     |     20 | 2975 |          800 |         3000 |
| Anil      |     20 | 2645 |          800 |         3000 |
| Kiran     |     20 | 1100 |          800 |         3000 |
| Prachi    |     20 | 3000 |          800 |         3000 |
| Amit      |     30 | 1600 |         1250 |         3000 |
| Rajesh    |     30 | 1250 |         1250 |         3000 |
| chaitanya |     30 | 1250 |         1250 |         3000 |
| Suresh    |     30 | 2850 |         1250 |         3000 |
| Rohit     |     30 | 1500 |         1250 |         3000 |
| Aparna    |     30 | 3000 |         1250 |         3000 |
+-----------+--------+------+--------------+--------------+
15 rows in set (0.0228 sec)
```

Shows min & max salary in each department next to every employee.

Department-wise Rank

```
15 rows in set (0.0228 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, deptno, sal,
                                          ->        RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS dept_rank
                                          -> FROM emp;
+-----------+--------+------+-----------+
| ename     | deptno | sal  | dept_rank |
+-----------+--------+------+-----------+
| Arjun     |   NULL | 4000 |         1 |
| Rakesh    |     10 | 5000 |         1 |
| Krishna   |     10 | 2450 |         2 |
| jahnavi   |     10 | 1300 |         3 |
| Prachi    |     20 | 3000 |         1 |
| Bujji     |     20 | 2975 |         2 |
| Anil      |     20 | 2645 |         3 |
| Kiran     |     20 | 1100 |         4 |
| Ammu      |     20 |  800 |         5 |
| Aparna    |     30 | 3000 |         1 |
| Suresh    |     30 | 2850 |         2 |
| Amit      |     30 | 1600 |         3 |
| Rohit     |     30 | 1500 |         4 |
| Rajesh    |     30 | 1250 |         5 |
| chaitanya |     30 | 1250 |         5 |
+-----------+--------+------+-----------+
15 rows in set (0.1287 sec)
```

Each department's employees ranked by salary.

Set Operators :

Used To Combine two queries we need set operators.

Types of Set Operators

- Union
- Union all
- Intersect
- Minus

SET OPERATORS IN SQL

Set operators are used to combine the results of two or more SELECT queries.

Each SELECT query must return the same number of columns with compatible data types.

Example Table: STUDENT

STUD_ID	NAME	COURSE
101	Arjun	Python
102	Neha	Java
103	Rahul	SQL
104	Kiran	HTML

Table: COURSE_APPLIED

STUD_ID	NAME	COURSE
201	Neha	Java
202	Kiran	HTML
203	Riya	Python
204	Suresh	C++

1. UNION

Description:

Combines results from both queries and removes duplicates.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT COURSE FROM STUDENT
                                         -> UNION
                                         -> SELECT COURSE FROM COURSE_APPLIED;
+--------+
| COURSE |
+--------+
| Python |
| Java   |
| SQL    |
| HTML   |
| C++    |
+--------+
```

Duplicate “Python” and “HTML” appear only once.

2. UNION ALL

Description:

Combines results from both queries **including duplicates**.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT COURSE FROM STUDENT
                                         -> UNION ALL
                                         -> SELECT COURSE FROM COURSE_APPLIED;
+--------+
| COURSE |
+--------+
| Python |
| Java   |
| SQL    |
| HTML   |
| Java   |
| HTML   |
| Python |
| C++    |
+--------+
```

Shows **all rows**, even duplicates.

3. INTERSECT

Description:

Returns **only common rows** between two queries.

```
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT COURSE FROM STUDENT
                                         -> INTERSECT
                                         -> SELECT COURSE FROM COURSE_APPLIED;
+--------+
| COURSE |
+--------+
| Python |
| Java   |
| HTML   |
+--------+
```

Only courses present in **both tables**.

4. MINUS / EXCEPT

Description:

Returns rows from **first query** that are **not in the second query**.

```
3 rows in set (0.0100 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT COURSE FROM STUDENT
                                         -> EXCEPT
                                         -> SELECT COURSE FROM COURSE_APPLIED;
+--------+
| COURSE |
+--------+
| SQL    |
+--------+
1 row in set (0.0016 sec)
```

JOINS

1. **Purpose:**

   The main purpose of a **JOIN** is to **combine data from two or more tables** based on a related column between them.

2. **How it works:**

   A join is performed using a **WHERE clause** (or **ON clause**) that specifies the condition to match rows from different tables.

3. **Multiple Table Joins:**

   If a join involves **more than two tables**, Oracle first joins the **first two tables** based on the join condition, and then joins the **resulting data** with the **next table**, continuing this process until all tables are joined.

Important Joins (Must Learn – used in real projects)

These are the **main and practical joins** you'll use in MySQL and almost every SQL job or project:

Join Type	Importance	Notes
INNER JOIN	★★★★	Most commonly used — gets matching rows.
LEFT JOIN (LEFT OUTER)	★★★★	Gets all from left + matching from right.
RIGHT JOIN (RIGHT OUTER)	★★★	Used less often, but still good to know.
FULL JOIN (FULL OUTER)	★★	Not directly supported in MySQL, but can be done using UNION.
CROSS JOIN	★	Rare, but useful to understand Cartesian products.
SELF JOIN	★★	Used in hierarchical or manager-employee type data.

If you know all the above — you can easily handle **90% of real-world SQL joins**.

JOIN: Used to combine data from two or more tables based on a related column.

1. INNER JOIN

Definition: Returns only the rows that have matching values in both tables.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename, d.dname
                                           -> FROM emp e
                                           -> INNER JOIN dept d
                                           -> ON e.deptno = d.deptno;

+-----------+--------------------+
| ename     | dname              |
+-----------+--------------------+
| Ammu      | ANALYST            |
| Amit      | Technical Trainer  |
| Rajesh    | Technical Trainer  |
| Bujji     | ANALYST            |
| chaitanya | Technical Trainer  |
| Suresh    | Technical Trainer  |
| Krishna   | python_developer   |
| Anil      | ANALYST            |
| Rakesh    | python_developer   |
| Rohit     | Technical Trainer  |
| Kiran     | ANALYST            |
| Aparna    | Technical Trainer  |
| Prachi    | ANALYST            |
| jahnavi   | python_developer   |
+-----------+--------------------+
```

Output: Only matching rows from both tables.

2. LEFT JOIN

Definition: Returns all rows from the left table, and matching rows from the right.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename, d.dname
                                           -> FROM emp e
                                           -> LEFT JOIN dept d
                                           -> ON e.deptno = d.deptno;

+-----------+--------------------+
| ename     | dname              |
+-----------+--------------------+
| Arjun     | NULL               |
| Ammu      | ANALYST            |
| Amit      | Technical Trainer  |
| Rajesh    | Technical Trainer  |
| Bujji     | ANALYST            |
| chaitanya | Technical Trainer  |
| Suresh    | Technical Trainer  |
| Krishna   | python_developer   |
| Anil      | ANALYST            |
| Rakesh    | python_developer   |
| Rohit     | Technical Trainer  |
| Kiran     | ANALYST            |
| Aparna    | Technical Trainer  |
| Prachi    | ANALYST            |
| jahnavi   | python_developer   |
+-----------+--------------------+
15 rows in set (0.0052 sec)
```

Out put :: All from left + matching right hear left table is emp so we get Arjun left join we give all rows from left table from right only matching

RIGHT JOIN

Definition: Returns all rows from the right table, and matching rows from the left.

Ex:SELECT e.ename, d.dname

FROM employee e

RIGHT JOIN department d

ON e.deptno = d.deptno;

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename, d.dname
                                           -> FROM employee e
                                           -> RIGHT JOIN department d
                                           -> ON e.deptno = d.deptno;
+-----------+--------------------+
| ename     | dname              |
+-----------+--------------------+
| jahnavi   | python_developer   |
| Rakesh    | python_developer   |
| Krishna   | python_developer   |
| Prachi    | ANALYST            |
| Kiran     | ANALYST            |
| Anil      | ANALYST            |
| Bujji     | ANALYST            |
| Ammu      | ANALYST            |
| Aparna    | Technical Trainer  |
| Rohit     | Technical Trainer  |
| Suresh    | Technical Trainer  |
| chaitanya | Technical Trainer  |
| Rajesh    | Technical Trainer  |
| Amit      | Technical Trainer  |
| NULL      | MANAGER            |
+-----------+--------------------+
15 rows in set (0.0022 sec)
```

FULL JOIN

Definition: Returns all rows from both tables (matching or not).

Note: MySQL doesn't support FULL JOIN directly — use UNION.

```
15 rows in set (0.0022 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename, d.dname
                                           -> FROM employee e
                                           -> LEFT JOIN department d
                                           -> ON e.deptno = d.deptno
                                           -> UNION
                                           -> SELECT e.ename, d.dname
                                           -> FROM employee e
                                           -> RIGHT JOIN department d
                                           -> ON e.deptno = d.deptno;
+-----------+--------------------+
| ename     | dname              |
+-----------+--------------------+
| Arjun     | NULL               |
| Ammu      | ANALYST            |
| Amit      | Technical Trainer  |
| Rajesh    | Technical Trainer  |
| Bujji     | ANALYST            |
| chaitanya | Technical Trainer  |
| Suresh    | Technical Trainer  |
| Krishna   | python_developer   |
| Anil      | ANALYST            |
| Rakesh    | python_developer   |
| Rohit     | Technical Trainer  |
| Kiran     | ANALYST            |
| Aparna    | Technical Trainer  |
| Prachi    | ANALYST            |
| jahnavi   | python_developer   |
| NULL      | MANAGER            |
+-----------+--------------------+
```

SELF JOIN

Definition: A table joins with itself.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename AS Employee, m.ename AS Manager
                                            -> FROM employee e
                                            -> JOIN employee m
                                            -> ON e.mgr = m.empno;
+-----------+---------+
| Employee  | Manager |
+-----------+---------+
| Ammu      | Prachi  |
| Amit      | Suresh  |
| Rajesh    | Suresh  |
| Bujji     | Rakesh  |
| chaitanya | Suresh  |
| Suresh    | Rakesh  |
| Krishna   | Rakesh  |
| Anil      | Bujji   |
| Rohit     | Suresh  |
| Kiran     | Anil    |
| Aparna    | Suresh  |
| Prachi    | Bujji   |
| jahnavi   | Krishna |
+-----------+---------+
```

CROSS JOIN

Definition: Returns all possible combinations of both tables.

If 4 departments × 15 employees → 60 total rows.

Used rarely for combinations or testing.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e.ename, d.dname
                                            -> FROM employee e
                                            -> CROSS JOIN department d;
+-----------+------------------+
| ename     | dname            |
+-----------+------------------+
| Arjun     | MANAGER          |
| Arjun     | Technical Trainer |
| Arjun     | ANALYST          |
| Arjun     | python_developer |
| Ammu      | MANAGER          |
| Ammu      | Technical Trainer |
| Ammu      | ANALYST          |
| Ammu      | python_developer |
| Amit      | MANAGER          |
| Amit      | Technical Trainer |
| Amit      | ANALYST          |
| Amit      | python_developer |
| Rajesh    | MANAGER          |
| Rajesh    | Technical Trainer |
| Rajesh    | ANALYST          |
| Rajesh    | python_developer |
| Bujji     | MANAGER          |
| Bujji     | Technical Trainer |
| Bujji     | ANALYST          |
| Bujji     | python_developer |
| chaitanya | MANAGER          |
| chaitanya | Technical Trainer |
| chaitanya | ANALYST          |
| chaitanya | python_developer |
| Suresh    | MANAGER          |
```

A. INNER JOIN – Matching Rows Only

1. Display employee name, job, department name, and location.

   Ans :: SELECT e.ename, e.job, d.dname, d.loc

   FROM emp e

   INNER JOIN dept d

   ON e.deptno = d.deptno;

2. List all employees working in the 'SALES' department.

   SELECT e.ename, e.job, d.dname

   FROM emp e

   JOIN dept d

   ON e.deptno = d.deptno

   WHERE d.dname = 'SALES';

3. Show employees who work in 'ACCOUNTING' department and earn salary > 2000.

   SELECT e.ename, e.sal, d.dname

   FROM emp e

   JOIN dept d

   ON e.deptno = d.deptno

   WHERE d.dname = 'ACCOUNTING' AND e.sal > 2000;

4. Display department name and total number of employees working in each department.

   SELECT d.dname, COUNT(e.empno) AS total_employees

   FROM dept d

   LEFT JOIN emp e

   ON d.deptno = e.deptno

   GROUP BY d.dname;

5. Find employees whose department location is 'CHICAGO'.

   SELECT e.ename, e.job, d.loc

   FROM emp e

   JOIN dept d

   ON e.deptno = d.deptno

   WHERE d.loc = 'CHICAGO';

LEFT JOIN – All EMP, matching DEPT

6. Display all employees along with their department names — show NULL if not assigned to any department.

```
SELECT e.ename, d.dname
FROM emp e
LEFT JOIN dept d
ON e.deptno = d.deptno;
```

7. Show employees who are **not assigned** to any department.

```
SELECT e.ename, e.deptno
FROM emp e
LEFT JOIN dept d
ON e.deptno = d.deptno
WHERE d.deptno IS NULL;
```

B. RIGHT JOIN – All DEPT, matching EMP

1. List all departments and their employees (show NULL for departments with no employees).

```
SELECT d.dname, e.ename
FROM emp e
RIGHT JOIN dept d
ON e.deptno = d.deptno;
```

2. Find departments that currently have **no employees**.

```
SELECT d.dname
FROM emp e
RIGHT JOIN dept d
ON e.deptno = d.deptno
WHERE e.empno IS NULL;
```

c). FULL OUTER JOIN (simulated with UNION)

All employees and departments (matched + unmatched)

```
SELECT e.ename, d.dname
FROM emp e
LEFT JOIN dept d
ON e.deptno = d.deptno
UNION
SELECT e.ename, d.dname
FROM emp e
RIGHT JOIN dept d
ON e.deptno = d.deptno;
```

SELF JOIN — Manager-Employee Relationship

1. Display employee name and their manager's name.

```
SELECT e.ename AS Employee, m.ename AS Manager
FROM emp e
LEFT JOIN emp m
ON e.mgr = m.empno;
```

2. Find employees who are managers (appear as someone's manager_id).

```
SELECT DISTINCT m.ename AS Manager
FROM emp e
JOIN emp m
ON e.mgr = m.empno;
```

3. Display employees who do not have a manager.

```
SELECT e.ename
FROM emp e
WHERE e.mgr IS NULL;
```

4. Show each manager and the count of employees working under them.

```
SELECT m.ename AS Manager, COUNT(e.empno) AS No_of_Employees
FROM emp e
JOIN emp m
ON e.mgr = m.empno
GROUP BY m.ename;
```

UPDATE and DELETE with JOIN

Increase salary by 10% for RESEARCH dept

```
UPDATE emp e
JOIN dept d
ON e.deptno = d.deptno
SET e.sal = e.sal * 1.10
WHERE d.dname = 'RESEARCH';
```

Delete employees from OPERATIONS dept

```
DELETE e
FROM emp e
JOIN dept d
ON e.deptno = d.deptno
WHERE d.dname = 'OPERATIONS';
```

Set commission to 0 for SALES dept

```
UPDATE emp e
JOIN dept d
ON e.deptno = d.deptno
SET e.comm = 0
WHERE d.dname = 'SALES';
```

BONUS CHALLENGES

1. Display the highest-paid employee in each department.
2. Find employees who earn more than their manager.
3. Display departments where average salary > 2500.
4. Show department name and number of managers working in that department.
5. Give **bonus of 500** to employees whose job is 'SALESMAN' and department is 'SALES'.
6. Delete employees who **earn less than 1000**.
7. Display department name, **highest salary, lowest salary, and average salary**.
8. Display all departments and **count of employees** in each (use left join).
9. Show departments where **no employees earn more than 2500**.
10. Display employees with salary **between 1000 and 3000**, including department name.

SUB – QUERIES, CO – RELATED QUERIES and EXISTS clause

Sub Queries: If we write select statement in where clause that we called it as sub queries or inner queries.

Types

WHERE CAN WE WRITE A SUB-QUERY IN SQL?

A **sub-query** can be written **in different parts** of a main (outer) SQL query:

WHERE clause

HAVING clause

FROM clause

SELECT clause

A **subquery** is a **SELECT statement written inside another SQL query**.

It is usually written in the **WHERE**, **HAVING**, or **FROM** clause of the main (outer) query.

The purpose of a sub-query is to **provide data** that the outer query can use.

1. Single Row Sub – Queries
2. Multi Row Sub – Queries
3. Correlated Sub – Queries

Types of Sub-Queries

Single Row Sub–Queries

A single-row sub-query is a sub-query that returns only one row and one column as output.

It is used with single-value comparison operators like

Used with operators: =, <, >, <=, >=, <>.

Find employees whose salary is the highest;

```
MySQL localhost:33060+ ssl 10kcoders SQL > rename table employee to emp;
Query OK, 0 rows affected (0.2724 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > -- Find employee(s) whose salary is the highest
Query OK, 0 rows affected (0.0015 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT ename, sal
                                        -> FROM emp
                                        -> WHERE sal = (SELECT MAX(sal) FROM emp);

+--------+------+
| ename  | sal  |
+--------+------+
| Rakesh | 5000 |
+--------+------+
1 row in set (0.0796 sec)
```

Step-by-Step Explanation

Step 1: Inner Query Execution

SELECT MAX(sal) FROM emp;

- This query runs **first** because it is the **inner query**.
- It checks all the values in the sal column of the **EMP** table.
- It calculates the **maximum salary** among all employees.
- Example result:

  MAX(sal)

  5000

So, the inner query returns **only one value** — the **highest salary (5000)**.

That's why this is called a **Single Row Sub-Query**.

Step 2: Outer Query Execution

SELECT ename, sal

FROM emp

WHERE sal = 5000;

- SQL now substitutes the result of the inner query (5000) into the outer query.
- The **outer query** goes through each row of the **EMP** table and checks:
  - "Is this employee's salary equal to 5000?"
- If **YES**, that employee is displayed in the result.
- If **NO**, that employee is skipped.

Step 3: Final Output

Example output might look like this:

ENAME SAL

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

RAKESH 5000

So, **RAKESH** is the employee who has the highest salary.

Single Row Sub-Query Practice Questions

1. Find the employee(s) whose salary is equal to the minimum salary in the company.

```
1 row in set (0.0798 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal
                                           -> FROM emp
                                           -> WHERE sal = (SELECT MIN(sal) FROM emp);

+-------+-----+
| ename | sal |
+-------+-----+
| Ammu  | 800 |
+-------+-----+
1 row in set (0.0038 sec)
```

2) Find the employee(s) who joined on the earliest date

Hint:

- You need to find the **minimum hire date** first using MIN(hiredate).
- Then select employees whose hiredate equals that date.

Use a **single row sub-query** in the WHERE clause.

3) Find all employees whose salary is equal to the salary of employee ‘Aparna’

Hint:

- First, get **Aparna’s salary** using a sub-query.
- Then find all employees having the **same salary**.

Use WHERE sal = (SELECT sal FROM emp WHERE ename = 'Aparna').

4)Find employee(s) whose commission is equal to the maximum commission given

Hint:

- Inner query: get the **maximum commission** using MAX(comm).
- Outer query: find employees whose comm matches that value.

Similar logic as highest salary example.

5)Find the department number which has the highest average salary

Hint:

- First, use GROUP BY deptno and AVG(sal) in a sub-query to find **average salary per department**.
- Then compare each department's average to the **maximum of those averages**.

You'll need a nested sub-query here (a query inside another sub-query).

6) List employees whose salary is exactly equal to the average salary of the company

Hint:

- Inner query: find company's **average salary** using AVG(sal).
- Outer query: select employees whose sal equals that average.

Single row sub-query (since AVG gives one value).

7) Find the employee(s) with the second highest salary

Hint:

- Inner query: get the **maximum salary** from EMP.
- Outer query: get the **maximum salary less than that maximum**.

You'll use < (SELECT MAX(sal) FROM emp) inside another MAX() function.

8) Find the employee(s) working in the department with the minimum dept number

Hint:

- Inner query: get **minimum deptno** from DEPT using MIN(deptno).
- Outer query: select employees whose deptno equals that value.

Simple single row sub-query.

9)Find the name(s) of employee(s) who report to the manager with the highest salary

Hint:

- Step 1: Find the **manager (mgr)** who has the **maximum salary**.
- Step 2: Find employees whose mgr = that manager's empno.

You'll use a sub-query that returns the empno of the manager with max salary

2. Multi Row Sub-Query

1. Returns **multiple rows**.
2. Used with operators: IN, ANY, ALL.

1.)Find employees who work in departments 10 or 20

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, deptno
                                           -> FROM emp
                                           -> WHERE deptno IN (SELECT deptno FROM dept WHERE deptno < 30);
+---------+--------+
| ename   | deptno |
+---------+--------+
| Ammu    |     20 |
| Bujji   |     20 |
| Krishna |     10 |
| Anil    |     20 |
| Rakesh  |     10 |
| Kiran   |     20 |
| Prachi  |     20 |
| jahnavi |     10 |
+---------+--------+
8 rows in set (0.0200 sec)
```

Step 1: Focus on the Inner Query

SELECT deptno FROM dept WHERE deptno IN (10, 20);

1. This query looks at the **DEPT** table.
2. It checks which department numbers are **10 or 20**.
3. It returns:

deptno

10

20

So, the inner query result = **{10, 20}**

Step 2: Outer Query Execution

SELECT ename, deptno

FROM emp

WHERE deptno IN (10, 20);

Now SQL takes the result of the inner query {10, 20}

and substitutes it in the outer query

So effectively, it becomes:

```
SELECT ename, deptno
FROM emp
WHERE deptno IN (10, 20);
```

1. The **EMP** table is scanned row by row.
2. For each employee, SQL checks:
3. Does this employee's deptno exist in {10, 20}?
4. If **YES**, that employee's record is included in the final result.
5. If **NO**, that employee's record is ignored.

Step 3: Output

You'll get a list of employees whose department number is **10 or 20** —
for example: out put

```
+---------+--------+
| ename   | deptno |
+---------+--------+
| Ammu    |     20 |
| Bujji   |     20 |
| Krishna |     10 |
| Anil    |     20 |
| Rakesh  |     10 |
| Kiran   |     20 |
| Prachi  |     20 |
| jahnavi |     10 |
+---------+--------+
```

Multi Row Sub-Query Practice Questions

1. Find employees who work in department 10 or 20.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, deptno
                                          -> FROM emp
                                          -> WHERE deptno IN (SELECT deptno FROM dept WHERE deptno IN (10, 20));
+---------+--------+
| ename   | deptno |
+---------+--------+
| Ammu    |     20 |
| Bujji   |     20 |
| Krishna |     10 |
| Anil    |     20 |
| Rakesh  |     10 |
| Kiran   |     20 |
| Prachi  |     20 |
| jahnavi |     10 |
+---------+--------+
8 rows in set (0.0203 sec)
```

Step 1: Focus on the Inner Query

```
SELECT deptno FROM dept WHERE deptno IN (10, 20);
```

1. This query looks at the **DEPT** table.
2. It checks which department numbers are **10 or 20**.
3. It returns:

2. Find employees who earn more than any employee in department 20.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT ename, sal, deptno
                                           -> FROM emp
                                           -> WHERE sal > ANY (SELECT sal FROM emp WHERE deptno = 20);
+-----------+------+--------+
| ename     | sal  | deptno |
+-----------+------+--------+
| Arjun     | 4000 |   NULL |
| Amit      | 1600 |     30 |
| Rajesh    | 1250 |     30 |
| Bujji     | 2975 |     20 |
| chaitanya | 1250 |     30 |
| Suresh    | 2850 |     30 |
| Krishna   | 2450 |     10 |
| Anil      | 2645 |     20 |
| Rakesh    | 5000 |     10 |
| Rohit     | 1500 |     30 |
| Kiran     | 1100 |     20 |
| Aparna    | 3000 |     30 |
| Prachi    | 3000 |     20 |
| jahnavi   | 1300 |     10 |
+-----------+------+--------+
```

3) Find employees who work in departments located in the city 'NEW YORK'

Hint:

- Inner query: get all department numbers where loc = 'NEW YORK' from the **DEPT** table.
- Outer query: select employees whose deptno is **in** that list.

  Use IN operator because multiple departments can exist in NEW YORK.

4)Find employees whose salary is in the list of all salaries of department 10

Hint:

- Inner query: select all sal values from **EMP** where deptno = 10.
- Outer query: select employees whose sal is **in that list**.

  Use IN operator since inner query returns multiple salaries.

5)List employees who do not work in departments 30 or 40

Hint:

- Inner query: select dept numbers (30, 40).
- Outer query: find employees whose deptno **not in** that list.

  Use NOT IN operator.

6)Find employees who earn less than all employees in department 20

Hint:

- Inner query: get all sal values from dept 20.
- Outer query: select employees whose salary < ALL of those salaries.

  Use ALL operator — means salary less than **every** value from dept 20.

7) Find employees whose manager is in department 10 or 30

Hint:

- Inner query: select all empno values of employees who work in dept 10 or 30 (these are potential managers).
- Outer query: find employees whose mgr value is **in** that list.

  Use IN operator because multiple manager IDs will be returned.

8) Find employees working in departments where average salary is more than 2000

Hint:

- Inner query: use GROUP BY deptno and HAVING AVG(sal) > 2000 to get dept numbers.
- Outer query: select employees whose deptno **in that list**.

  Use IN operator — multiple departments may satisfy the condition.

9) Find employees whose salary matches salaries of any employees earning above 2500

Hint:

- Inner query: get all salaries from employees earning > 2500.
- Outer query: find employees whose salary = ANY of those salaries.

  Use ANY operator — means match any one of the returned salary values.

10) Find employees who earn more than any employee working as a 'CLERK'.

Hint:

- Inner query: select all sal values from **EMP** where job = 'CLERK'.
- Outer query: select employees whose salary > ANY of those salaries.

  Use the ANY operator — it means the employee's salary is **greater than at least one CLERK's salary**.

Conclusion / Summary of Multi Row Sub-Query

1. A **Multi Row Sub-Query** is a sub-query that returns **more than one row** of results.
2. Because it returns multiple values, we **cannot** use simple comparison operators like = or <.
3. Instead, we use **multi-value comparison operators**:
   a) IN → Checks if a value matches **any value** in the list.
   b) ANY → Compares with **at least one** value from the sub-query.
   c) ALL → Compares with **all** values from the sub-query.
4. The **inner query** executes first and provides a set of values to the **outer query**.
5. Multi Row Sub-Queries are useful when we need to compare data **between multiple rows** or **between different groups** of a table.

What is a Correlated Sub-Query?

A **Correlated Sub-Query** is a type of sub-query that **depends on the outer query** for its values.

In this case, the **inner query is executed once for every row** processed by the outer query.

Key Points:

- The **inner query** uses columns from the **outer query**.
- It **cannot run independently** because it depends on the outer query's values.
- Usually used with keywords like EXISTS, NOT EXISTS, IN, or comparison operators.

Correlated Sub-Query Questions

1)Find employees whose salary is greater than the average salary of their department.

Query:

```
8 rows in set (0.0176 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT e1.ename, e1.sal, e1.deptno
                                           -> FROM emp e1
                                           -> WHERE e1.sal > (
                                           ->     SELECT AVG(e2.sal)
                                           ->     FROM emp e2
                                           ->     WHERE e1.deptno = e2.deptno
                                           -> );
+--------+------+--------+
| ename  | sal  | deptno |
+--------+------+--------+
| Bujji  | 2975 |     20 |
| Suresh | 2850 |     30 |
| Anil   | 2645 |     20 |
| Rakesh | 5000 |     10 |
| Aparna | 3000 |     30 |
| Prachi | 3000 |     20 |
+--------+------+--------+
```

Explanation:

a) The outer query picks an employee e1.

b) The inner query calculates the average salary of that employee's department.

c) Then it checks whether the employee's salary is greater than that department's average.

d) This process repeats for every employee — that's why it's a correlated sub-query.

Step-by-Step Explanation

Outer Query (e1):

SELECT e1.ename, e1.sal, e1.deptno

FROM emp e1

- This part goes through **each employee** (row by row) in the emp table.
- e1 is just an **alias** for the outer query's employee table.

Inner Query (e2):

SELECT AVG(e2.sal)

FROM emp e2

WHERE e1.deptno = e2.deptno

- For **each employee from the outer query (e1)**, this sub-query runs separately.
- It calculates the **average salary of that employee's department**.
- Notice the condition:
- WHERE e1.deptno = e2.deptno

This connects the **outer query (e1)** with the **inner query (e2)** —

that's why this is called a **correlated sub-query**.

- The inner query depends on the current row of the outer query.

WHERE Condition:

WHERE e1.sal > (sub-query result)

- After the inner query calculates the **average salary** for that department,
  SQL checks whether the current employee's salary (e1.sal) is **greater than** that average.

How it Executes:

Let's say your table has these rows (simplified example):

ENAME	SAL	DEPTNO
Asha	1000	10
Ramesh	2000	10
Sita	3000	10
Arjun	1500	20
Meena	2500	20

👉 SQL processes each row like this:

- For **Asha (Dept 10)**: inner query finds avg(sal) in dept 10 = (1000+2000+3000)/3 = 2000 → 1000 > 2000 ❌
- For **Ramesh (Dept 10)**: 2000 > 2000 ❌
- For **Sita (Dept 10)**: 3000 > 2000 ✅

- For **Arjun (Dept 20)**: avg(Dept 20) = (1500+2500)/2 = 2000 → 1500 > 2000 ❌
- For **Meena (Dept 20)**: 2500 > 2000 ✅

✅ Final Output: **Sita, Meena**

Why It's Correlated

- Because the inner query **depends** on the current outer query row (e1.deptno).
- If you try to run the inner query alone, it will show an error —
  it **needs** a value from the outer query to work.

In Simple Words:

"For each employee, find the average salary of their department.

If the employee's salary is higher than that average, show them."

2) Find employees who earn more than their manager.

Hint: Compare employee salary with manager's salary using mgr column.

Inner query selects the manager's salary where empno = outer.mgr.

3) Find departments where at least one employee earns more than 3000.

Hint: Use EXISTS with inner query checking for sal > 3000 in the same dept.

```
6 rows in set (0.0055 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT d.deptno, d.dname
                                          -> FROM dept d
                                          -> WHERE EXISTS (
                                          ->     SELECT 1
                                          ->     FROM emp e
                                          ->     WHERE e.deptno = d.deptno
                                          ->       AND e.sal > 3000
                                          -> );
+--------+------------------+
| deptno | dname            |
+--------+------------------+
|     10 | python_developer |
+--------+------------------+
1 row in set (0.0025 sec)
```

Step-by-Step Explanation

Outer Query (dept d)

SELECT d.deptno, d.dname

FROM dept d

- Loops through **each department** in the DEPT table.
- Goal: return departments that have **at least one high-earning employee**.

Inner Query (emp e)

```
SELECT 1
FROM emp e
WHERE e.deptno = d.deptno
  AND e.sal > 3000
```

- Checks if there is **any employee in this department** with sal > 3000.
- SELECT 1 just needs to return a row — actual values don't matter for EXISTS.

How it Executes with Your EMP Data

Departments and employees:

- **Dept 10:** Krishna (2450), Rakesh (5000), jahnavi (1300) → Rakesh > 3000 ☑
- **Dept 20:** Ammu (800), Bujji (2975), Anil (2645), Kiran (1100), Prachi (3000) → Prachi = 3000 ☑
- **Dept 30:** Amit (1600), Rajesh (1250), chaitanya (1250), Suresh (2850), Rohit (1500), Aparna (3000) → Aparna = 3000 ☑

Note: Depending on strict > 3000, only Dept 10 has employee **Rakesh (5000)** > 3000.

Result

```
DEPTNO
-------
10
```

Only **Department 10** satisfies the condition.

Why it's a Correlated Sub-Query

- Inner query uses d.deptno from the outer query.
- It executes **once for each department** in DEPT.
- That dependency makes it **correlated**.

4)Find employees whose commission is greater than the average commission of their department.

Hint: Similar to Q1 but use comm instead of sal.

5)Find employees who have the same job title as someone in department 10.

Hint: Use EXISTS and compare job column of outer and inner query with deptno = 10.

6)Find departments that have no employees.

Hint: Use NOT EXISTS with a sub-query on emp table where dept.deptno = emp.deptno.

7) Find employees whose salary is greater than at least one employee in department 20.

Hint: Use > ANY with correlated condition between outer and inner emp tables.

8)Find employees who are the highest paid in their department.

Hint: Check if **no one else in that department** has a higher salary → use NOT EXISTS.

9)Find employees who joined before all employees in the same department.

Hint: Compare hiredate using correlated sub-query with deptno condition.

10Find employees who don't have any subordinates.

Hint: Use NOT EXISTS where inner query checks if anyone's mgr = outer.empno.

Conclusion:

Correlated Sub-Queries execute **row by row**, where the inner query depends on the outer query for its data. They are slower than normal sub-queries but very useful for **row-wise comparisons** and **complex filtering** using EXISTS or NOT EXISTS.

Nested Sub-Query?

A **Nested Sub-Query** is a sub-query that **contains another sub-query inside it**.

- Essentially, it's a **sub-query within a sub-query**, forming a **multi-level query**.
- The **innermost query** executes first, then its result is used by the **next outer query**, and finally the **outermost query** uses that result to produce the final output.

This is a classic **nested sub-query** problem:

Key Points:

1. A nested sub-query is often used when you need **stepwise filtering or computation**.
2. It allows you to **solve complex queries in parts**, like "find the second highest salary" or "find employees in the department with the highest average salary."
3. Can be **single-row or multi-row** at any level.
4. Each level depends on the **inner query's result** to feed the next query.

Innermost query → executed first

Middle query → uses innermost query's result

Outer query → final selection using middle query's result

To get the **3rd highest**, you need **two nested levels** below the highest, like this

```
1 row in set (0.0030 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT MAX(sal) AS Third_Highest_Salary
                                          -> FROM emp
                                          -> WHERE sal < (
                                          ->     SELECT MAX(sal)
                                          ->     FROM emp
                                          ->     WHERE sal < (
                                          ->         SELECT MAX(sal)
                                          ->         FROM emp
                                          ->     )
                                          -> );
+----------------------+
| Third_Highest_Salary |
+----------------------+
|                 3000 |
+----------------------+
1 row in set (0.0228 sec)
```

1. Innermost → 5000 (highest)
2. Middle → 4000 (2nd highest)
3. Outer → 3000 (3rd highest) ✅

Step 4: To get names of employees with 3rd highest salary

Conclusion / Notes on Nested Queries

1. **Definition:** A nested query is a **query inside another query**.
2. **Use Cases:**
   - Find **top N salaries**
   - Find **specific ranges of data**
   - Compare values against **aggregates** (MAX, MIN, AVG)
3. **Rules:**
   - Inner query executes **first**.
   - Outer query uses the result of the inner query.
   - You can **nest multiple levels** for 3rd, 4th, … Nth highest value.

VIEWS, INLINE VIEWS AND MATERIALIZED VIEWS

• A view is a database object that is a logical representation of a table.

• It is delivered from a table but has no storage of its on.

• View will fetch the data from base table.

It will run the Base query.

• A view takes the output of the query and treats it as a table Types ✓ Simple view ✓ Complex view

• Simple view can be created from one table where as complex view can be created from multiple tables.

• We can do DML on Simple view but not on Complex views.

Views in SQL

Definition:

A **View** is a **virtual table** in SQL. It doesn't store data physically. Instead, it displays data from one or more base tables using a SELECT query.

- Think of a view as a **window** to your data—it shows the data but doesn't store it.
- Whenever you query a view, SQL fetches the latest data from the underlying table(s).

Syntax to create a view:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > CREATE VIEW emp_view AS
                                           -> SELECT ename, sal, deptno
                                           -> FROM emp
                                           -> WHERE sal > 2000;
Query OK, 0 rows affected (0.0917 sec)
MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from emp_view;
+---------+------+--------+
| ename   | sal  | deptno |
+---------+------+--------+
| Arjun   | 4000 |   NULL |
| Bujji   | 2975 |     20 |
| Suresh  | 2850 |     30 |
| Krishna | 2450 |     10 |
| Anil    | 2645 |     20 |
| Rakesh  | 5000 |     10 |
| Aparna  | 3000 |     30 |
| Prachi  | 3000 |     20 |
+---------+------+--------+
8 rows in set (0.0262 sec)
```

It will show all employees with salary greater than 2000.

Types of Views

A.Simple View

Created from **one table only**.

Can perform **DML operations** (INSERT, UPDATE, DELETE) if it does not have aggregation or joins.

EX::

```
8 rows in set (0.0262 sec)
 MySQL  localhost:33060+ ssl  10kcoders  SQL > CREATE VIEW simple_emp AS
                                          -> SELECT ename, sal
                                          -> FROM emp
                                          -> WHERE deptno = 10;
Query OK, 0 rows affected (0.0162 sec)
 MySQL  localhost:33060+ ssl  10kcoders  SQL > SHOW TABLES;
+----------------------+
| Tables_in_10kcoders  |
+----------------------+
| course_applied       |
| dept                 |
| emp                  |
| emp1_bkp             |
| emp_bkp              |
| emp_empty            |
| emp_view             |
| simple_emp           |
| student              |
| telugu_movies        |
+----------------------+
10 rows in set (0.0085 sec)
```

You **can** update delete insert with view and it will update on main table also

```
Rows matched: 0  Changed: 0  Warnings: 0
 MySQL  localhost:33060+ ssl  10kcoders  SQL > UPDATE simple_emp SET sal = sal + 500 WHERE ename = 'krishna';
Query OK, 1 row affected (0.0205 sec)

Rows matched: 1  Changed: 1  Warnings: 0
 MySQL  localhost:33060+ ssl  10kcoders  SQL > select * from simple_emp;
+---------+------+
| ename   | sal  |
+---------+------+
| Krishna | 2950 |
| Rakesh  | 5000 |
| jahnavi | 1300 |
+---------+------+
3 rows in set (0.0213 sec)
```

B. Complex View

- Created from multiple tables using joins, aggregations, group by, etc.
- DML operations are not allowed directly.

```
 MySQL  localhost:33060+ ssl  10kcoders  SQL > CREATE VIEW complex_emp AS
                                          -> SELECT e.ename, e.sal, d.dname
                                          -> FROM emp e
                                          -> JOIN dept d ON e.deptno = d.deptno;
Query OK, 0 rows affected (0.0408 sec)
 MySQL  localhost:33060+ ssl  10kcoders  SQL >
```

You **cannot** do:

UPDATE complex_emp

SET sal = sal + 500;

It will give an error.

C.Inline Views

- An **inline view** is a **subquery used in the FROM clause**.
- It's not stored in the database, but acts as a temporary view for that query only.

```
MySQL  localhost:33060+ ssl  10kcoders  SQL > SELECT *
                                        -> FROM (SELECT ename, sal
                                        ->       FROM emp
                                        ->       WHERE sal > 2000) AS temp
                                        -> WHERE sal < 5000;
+---------+------+
| ename   | sal  |
+---------+------+
| Arjun   | 4000 |
| Bujji   | 2975 |
| Suresh  | 2850 |
| Krishna | 2950 |
| Anil    | 2645 |
| Aparna  | 3000 |
| Prachi  | 3000 |
+---------+------+
```

We can drop view with name:

```
 10kcoders  SQL > drop view complex_emp;
.0371 sec)
 10kcoders  SQL >
```

D.Materialized Views

- Unlike regular views, a **materialized view stores the query result physically** in the database.
- Useful for **reporting and performance**, because it doesn't recompute data every time.
- Can be **refreshed periodically** to get updated data.

```
> CREATE MATERIALIZED VIEW emp_mv
> AS
> SELECT deptno, AVG(sal) AS avg_salary
> FROM emp
> GROUP BY deptno;
```

Reason: MySQL does not support MATERIALIZED VIEW natively. That syntax works in Oracle or PostgreSQL, but not in MySQL.

In MySQL, you have to simulate a materialized view using either:

You create a **regular table** and populate it with a query result:

CREATE TABLE emp_mv AS

SELECT deptno, AVG(sal) AS avg_salary

FROM emp

GROUP BY deptno;

- This creates a **physical table** with the aggregated data.
- Whenever the base table changes, you need to **refresh manually**:

View Type	Base Tables	Storage	DML Allowed	Notes
Simple View	1	No	Yes	Single table, can update
Complex View	2+	No	No	Join/aggregation, read-only
Inline View	1+	No	No	Temporary for query only
Materialized View	1+	Yes	No	Use table + refresh manually

Indexes in MySQL

1. What is an Index?

Definition:

An **index** is a **database object** that improves the speed of data retrieval from a table. Think of it like an **index in a book**—it helps you find data faster without scanning the entire table.

- Index **does not store extra data** (except in certain types like FULLTEXT or unique indexes).
- Mainly used to **optimize SELECT queries**.
- **Downside:** It can slow down INSERT, UPDATE, and DELETE because the index also needs to be updated.

2. Why Use Indexes?

- To **speed up searching** in large tables.
- To **enforce uniqueness** (PRIMARY KEY or UNIQUE indexes).
- To **help join operations**.

Example: Without an index:

```
SELECT * FROM emp WHERE ename = 'Ammu';
```

- MySQL scans **every row** in emp → slow for large tables.

With an **index on ename**, MySQL can **jump directly to the matching rows** → much faster.

3. Types of Indexes in MySQL

Type	Description
PRIMARY KEY	Unique + Not NULL. Each table can have **one primary key**.
UNIQUE	Values must be unique. NULL allowed (only once for MySQL).
INDEX / KEY	Regular index to speed up queries.
FULLTEXT	For text search in CHAR, VARCHAR, TEXT columns.
SPATIAL	For spatial data types (geometry, point, etc.).

4. Creating Indexes

A. PRIMARY KEY

```
CREATE TABLE emp (
   empno INT PRIMARY KEY,
   ename VARCHAR(20),
   sal DECIMAL(10,2)
);
```

- Automatically creates a **unique index** on empno.

B. UNIQUE Index

```
CREATE UNIQUE INDEX idx_ename
ON emp(ename);
```

- Prevents duplicate employee names in the table.

C. Regular Index

```
CREATE INDEX idx_sal
ON emp(sal);
```

- Speeds up queries like:

```
SELECT * FROM emp
WHERE sal > 3000;
```

D. Composite Index

- Index on **multiple columns**.

```
CREATE INDEX idx_dept_sal
ON emp(deptno, sal);
```

- Speeds up queries like:

```
SELECT * FROM emp
WHERE deptno = 10 AND sal > 2000
```

5. Drop an Index

```
DROP INDEX idx_sal ON emp;
```

Quick Notes

- Indexes speed up SELECT, but slow down INSERT/UPDATE/DELETE.
- Use indexes only on columns frequently used in WHERE, JOIN, ORDER BY.
- MySQL automatically creates **indexes for PRIMARY KEY and UNIQUE constraints**.

TCL (Transaction Control Language) and DCL (Data Control Language) in MySQL

1. TCL (Transaction Control Language)

Definition:

TCL commands are used to **manage transactions** in the database. A transaction is a **group of SQL statements executed as a single unit**.

- If any statement fails, the whole transaction can be **rolled back**.
- If all statements succeed, the transaction can be **committed**.

Common TCL Commands in MySQL:

Command	Description
COMMIT	Saves all changes made in the transaction permanently.
ROLLBACK	Undoes all changes made in the current transaction.
SAVEPOINT	Creates a point within a transaction to roll back to partially.
SET TRANSACTION	Sets the properties of a transaction (like isolation level).

Example of TCL

```
Set auto
-- Example operations
UPDATE emp SET sal = sal + 500 WHERE empno = 1;
UPDATE emp SET sal = sal + 500 WHERE empno = 2;

-- If everything is fine, commit changes
COMMIT;

-- If something goes wrong, rollback
-- ROLLBACK;
```

Using SAVEPOINT:

```
START TRANSACTION;

UPDATE emp SET sal = sal + 500 WHERE  ename = 'AMMU';

SAVEPOINT before_second_update;
```

UPDATE emp SET sal = sal + 500 WHERE empno = 2;

-- Rollback only the second update

ROLLBACK TO SAVEPOINT before_second_update;

COMMIT;

Only the first update is saved permanently.

2. DCL (Data Control Language)

Definition:

DCL commands are used to **control access and permissions** for database users.

Common DCL Commands:

Command	Description
GRANT	Gives privileges to a user (like SELECT, INSERT, UPDATE).
REVOKE	Removes privileges from a user.

Examples of DCL

-- Create a user

CREATE USER 'john@localhost' IDENTIFIED BY 'john123';

-- Grant privileges

GRANT SELECT, INSERT, UPDATE ON testdb.* TO 'john@localhost';

-- Revoke privileges

REVOKE UPDATE ON testdb.* FROM 'john@localhost';

- GRANT → give permission
- REVOKE → remove permission

Quick Summary Table

Feature	TCL	DCL
Purpose	Manage transactions	Control user access
Commands	COMMIT, ROLLBACK, SAVEPOINT	GRANT, REVOKE
Affect Data	Yes (TCL changes data)	No (DCL controls permissions)
Example Usage	Commit salary updates	Grant user SELECT rights

TRIGGERS