

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

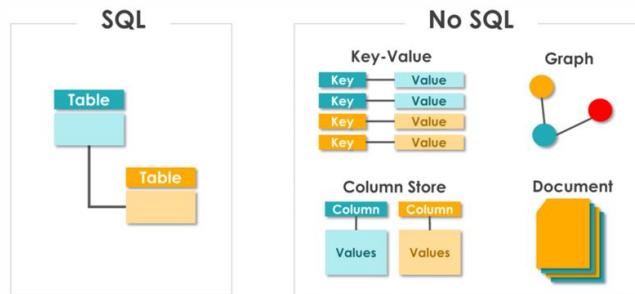
Relational databases:

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

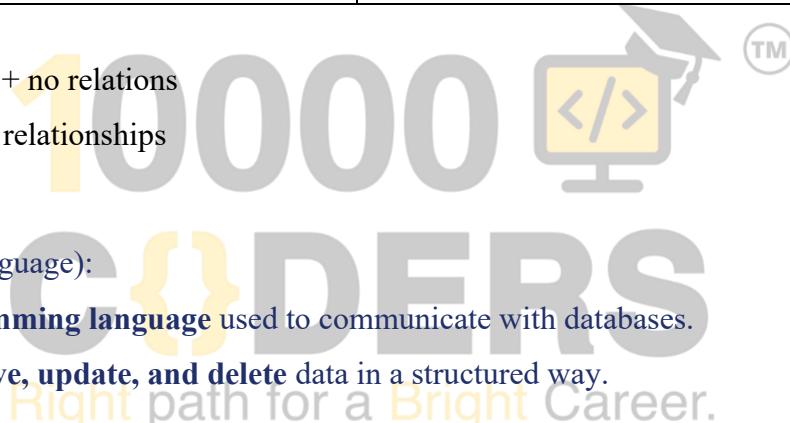
→ RDBMS = table-based + relationships

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.



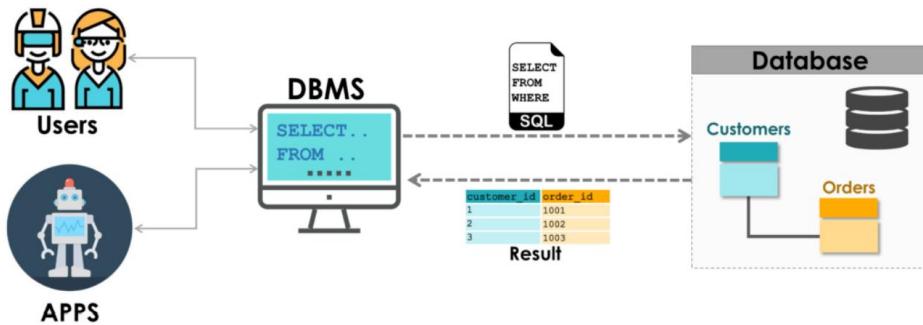
How does SQL Works?

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:

- Widely Used:** SQL is one of the most-used languages for working with databases.
- High Demand:** Developers with SQL skills are always in demand across industries.
- Universal Application:** SQL is used everywhere — from small businesses to big tech companies.
- Easy to Learn:** Compared to other programming languages, SQL is simple and beginner-friendly.
- 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

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

Origin

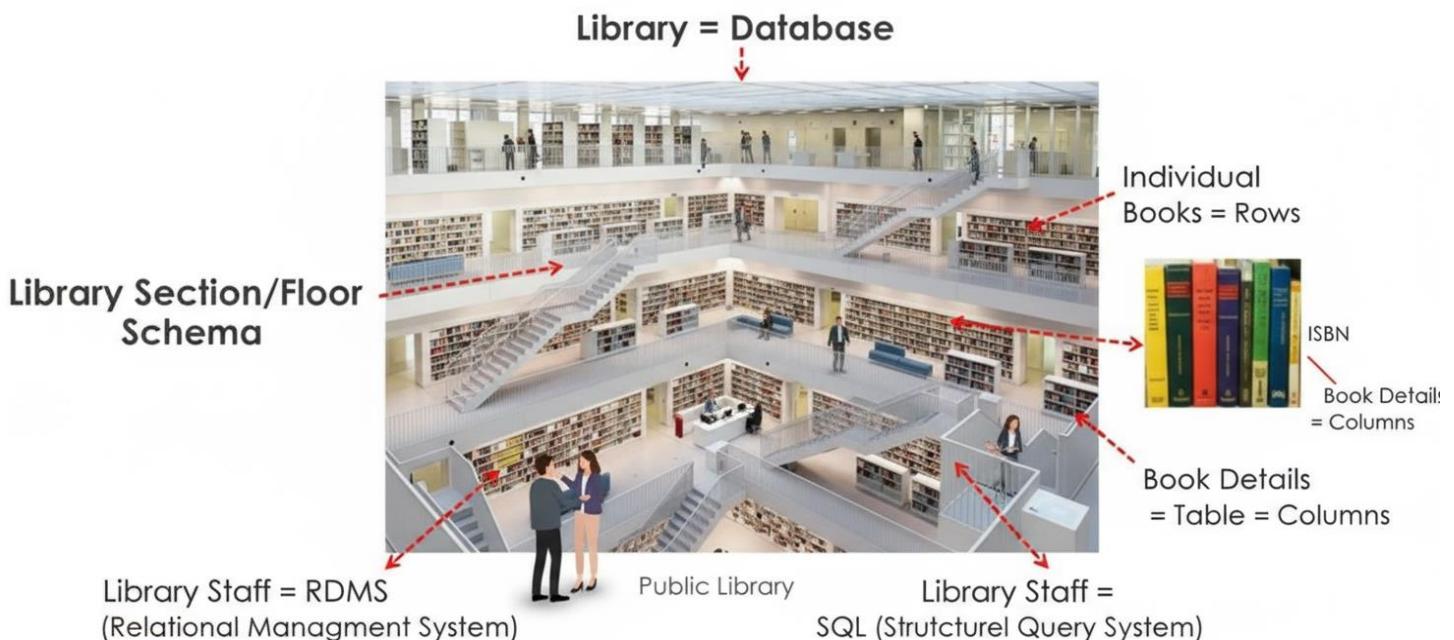
- In the **1970s**, IBM researchers **Donald D. Chamberlin** and **Raymond F. Boyce** developed a language called **SEQUEL (Structured English Query Language)**.
- It was designed to interact with IBM's **System R** — the first prototype of a **Relational Database Management System (RDBMS)** based on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

Evolution Timeline

Year	Event
1970	E. F. Codd, an IBM researcher, published a paper titled " <i>A Relational Model of Data for Large Shared Data Banks</i> ", 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.

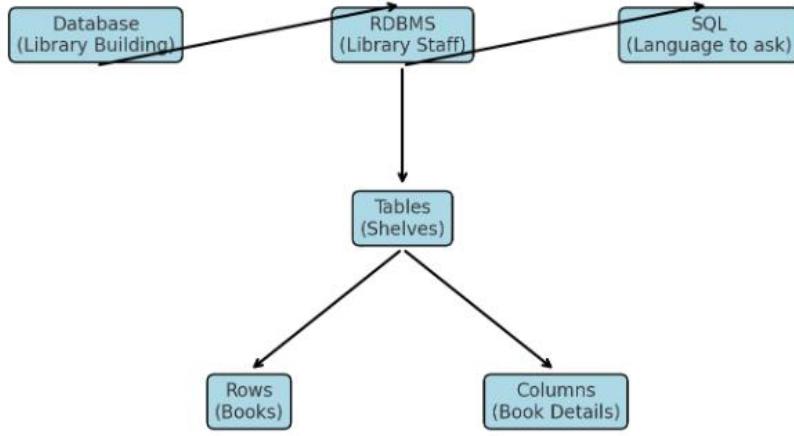
How is SQL Database organized?



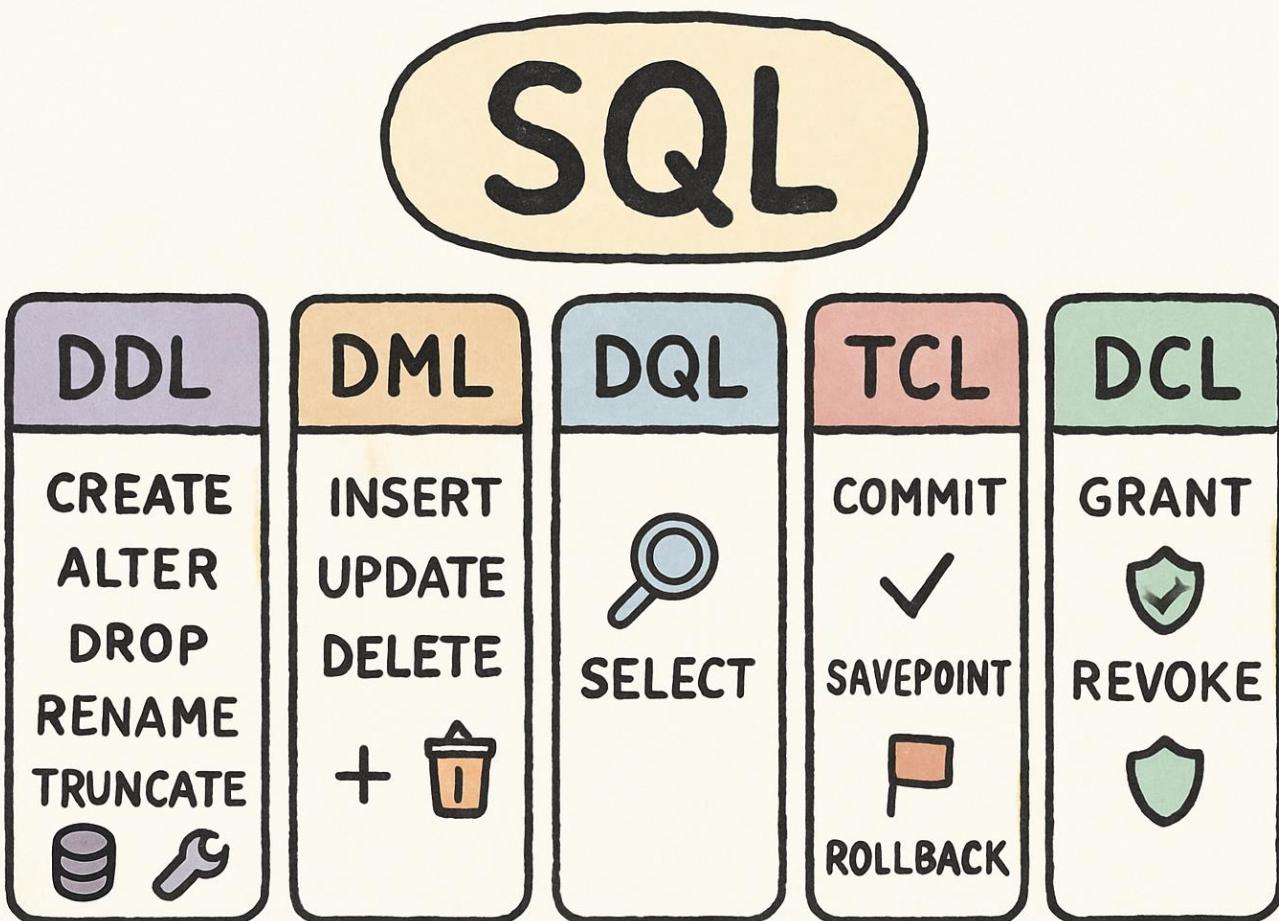
Library Analogy for Databases

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	$-2^{63} - 2^{63} - 2^{63}$	0	$2^{63} - 1$	$2^{64} - 1$

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

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

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

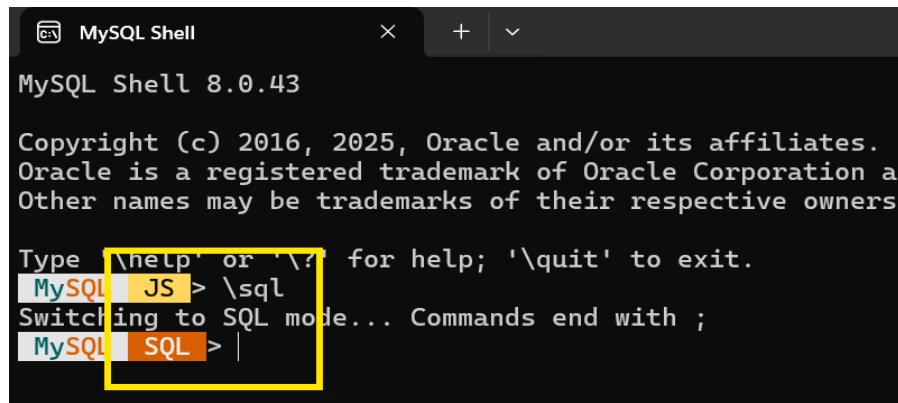
Database: A container that holds tables.

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`



The screenshot shows the MySQL Shell interface. It starts with the MySQL Shell 8.0.43 logo and copyright information. Then, it displays a message about switching modes. Finally, it shows the prompt changing from 'MySQL> JS>' to 'MySQL> SQL>'. The line 'MySQL> JS> \sql' is highlighted with a yellow box.

```

MySQL Shell 8.0.43

Copyright (c) 2016, 2025, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and
Other names may be trademarks of their respective owners.

Type '\help' or '\?' for help; '\quit' to exit.
MySQL> JS> \sql
Switching to SQL mode... Commands end with ;
MySQL> SQL>

```

Connect to MySQL Server:

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

```
MySQL | SQL > \c root@localhost:3306
Creating a session to 'root@localhost:3306'
Fetching global names for auto-completion... Press ^C to stop.
Your MySQL connection id is 40
Server version: 8.0.43 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL | localhost:3306 ssl SQL >
```

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
<code>\sql</code>	Switch to JS mode TO SQL in MySQL Shell
<code>\connect root@localhost:3306</code> or <code>\c root@localhost:3306</code>	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;

```
Query OK, 1 row affected (0.0312 sec)
MySQL | localhost:3306 ssl | SQL > create database 10kcoders;
Query OK, 1 row affected (0.0312 sec)
MySQL | localhost:3306 ssl | SQL > use 10kcoders;
Default schema set to '10kcoders'.
Fetching global names, object names from '10kcoders' for auto-completion
MySQL | localhost:3306 ssl | 10kcoders | SQL >
```

DDL (Data Definition Language)

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

1. CREATE

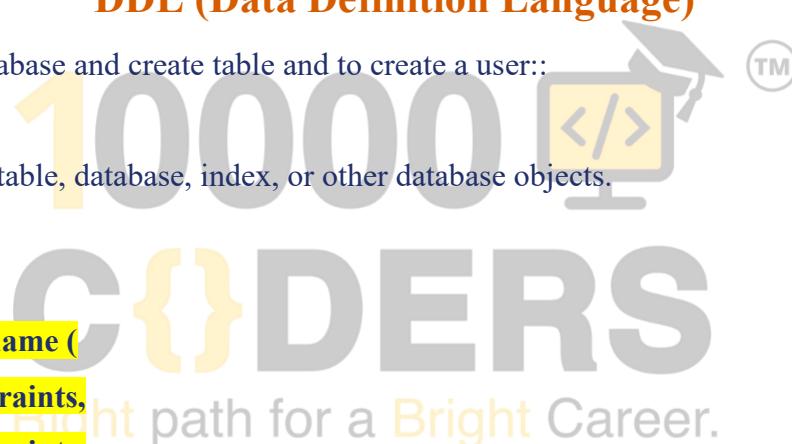
Description: Creates a new table, database, index, or other database objects.

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;**

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 Right path for a Bright Career.

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

```
/ 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;

```
, 1 rows in set (0.0237 sec)
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);

```
0 rows in set (0.0243 sec)
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);`

```

ERROR: 1146. Table '10kcoders.emp' doesn't exist
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);`



```

RECORDS: 0  Duplicates: 0  Warnings: 0
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 INTO table_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_name SET 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

```
+ rows in set (0.0014 sec)
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 rows in set (0.0012 sec)
```

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

```
+ row in set (0.0015 sec)
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 !=)

```
+ rows in set (0.0010 sec)
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.

```
+ rows in set (0.0010 sec)
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 |
+-----+-----+-----+-----+
```

Right path for a Bright Career.

2. Using OR

Returns rows if *any one* condition is true.

```
+ rows in set (0.0007 sec)
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)
```

Right path for a Bright Career.

5. Using IN

↳ Selects rows matching any value in a list.

```
5 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.

```
+ 3 rows in set (0.0013 sec)
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).

```
+ 0 rows in set (0.0003 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.



```
+ 5 rows in set (0.0004 sec)
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’

```
+ row in set (0.0000 sec)
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’

```
ERROR: 1065: query was empty
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

```
1 row in set (0.0001 sec)
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    |
+-----+-----+-----+-----+
```

Right path for a Bright Career.

2. Sort movies by year (oldest first)

```
5 rows in set (0.0015 sec)
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>;
```

```
ERROR: 1146. Table '10kcoders.student' doesn't exist
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

```
ERROR: query was empty
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.



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

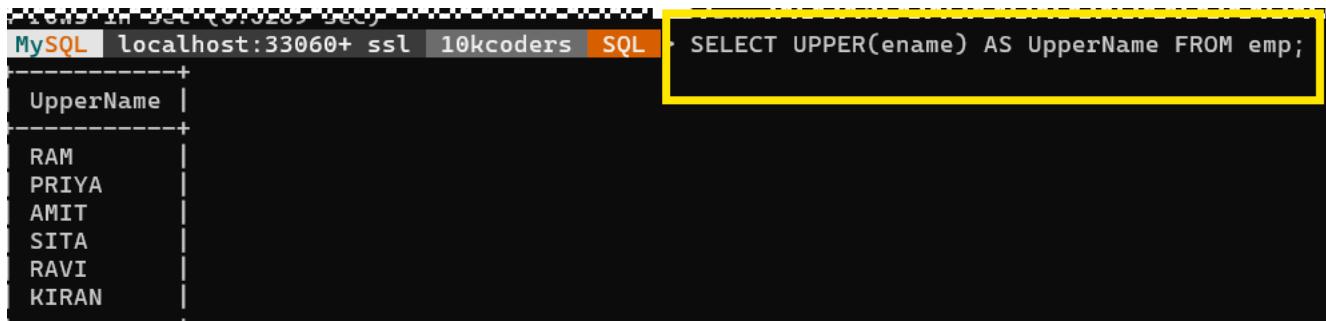
MySQL provides many functions to manipulate text/strings.

1] UPPER() / UCASE()

- Converts all letters to uppercase.



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



UpperName
RAM
PRIYA
AMIT
SITA
RAVI
KIRAN

2] LOWER() / LCASE()

Right path for a Bright Career.

- 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;
```

```
ERROR: 1054: Unknown column 'deptno' in 'field list'
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

```
ERROR: 1054: Unknown column 'sat' in 'field list'
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.

```
+6 rows in set (0.0021 sec)
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()

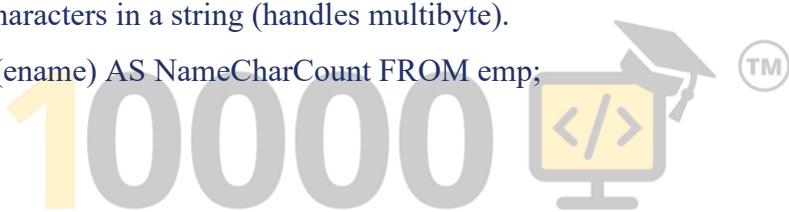
- A. Returns number of bytes in a string.

```
+-----+
| NameLength |
+-----+
| 3          |
| 5          |
| 4          |
| 4          |
| 4          |
| 5          |
+-----+
5 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).

```
+-----+
| TrimmedName |
+-----+
| RAM         |
| Priya       |
| Amit        |
| Sita        |
| Ravi        |
| Kiran       |
+-----+
```

-- Remove specific character

```
+-----+
| Cleaned   |
+-----+
| abc       |
+-----+
1 row in set (0.0006 sec)
```

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.

```
1 row in set (0.0000 sec)
MySQL localhost:33060+ ssl 10kcoders SQL > SELECT REPLACE(ename, 'A', '@') AS ReplacedName FROM emp;
+-----+
| ReplacedName |
+-----+
| R@M
| Priya
| @mit
| Sita
| Ravi
| Kiran
+-----+
```

1 1 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)
```

1 2 LOCATE()

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

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

1 ③ 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 |
+-----+
| iya
| mit
| ita
| avi
| ran
+-----+
6 rows in set (0.0014 sec)
```

1 ④ REVERSE()

- Reverses the string.



```
6 rows in set (0.0014 sec)
MySQL | localhost:33060+ ssl | 10kcoders | SQL > SELECT REVERSE(ename) AS ReversedName FROM emp;
+-----+
| ReversedName |
+-----+
| MAR
| ayirP
| timA
| atiS
| ivaR
| nariK
+-----+
```

1 ⑤ 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)
```

1 [6] 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)
```

1 [7] FORMAT() (for numbers, optional with strings)

- Formats numbers as strings with commas/decimals.

```
ERROR: 1054: Unknown column 'sat' in 'field list'
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.

```
1 row in set (0.0213 sec)
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.

```
1 row in set (0.0017 sec)
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
ASCII(s)	Returns 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'

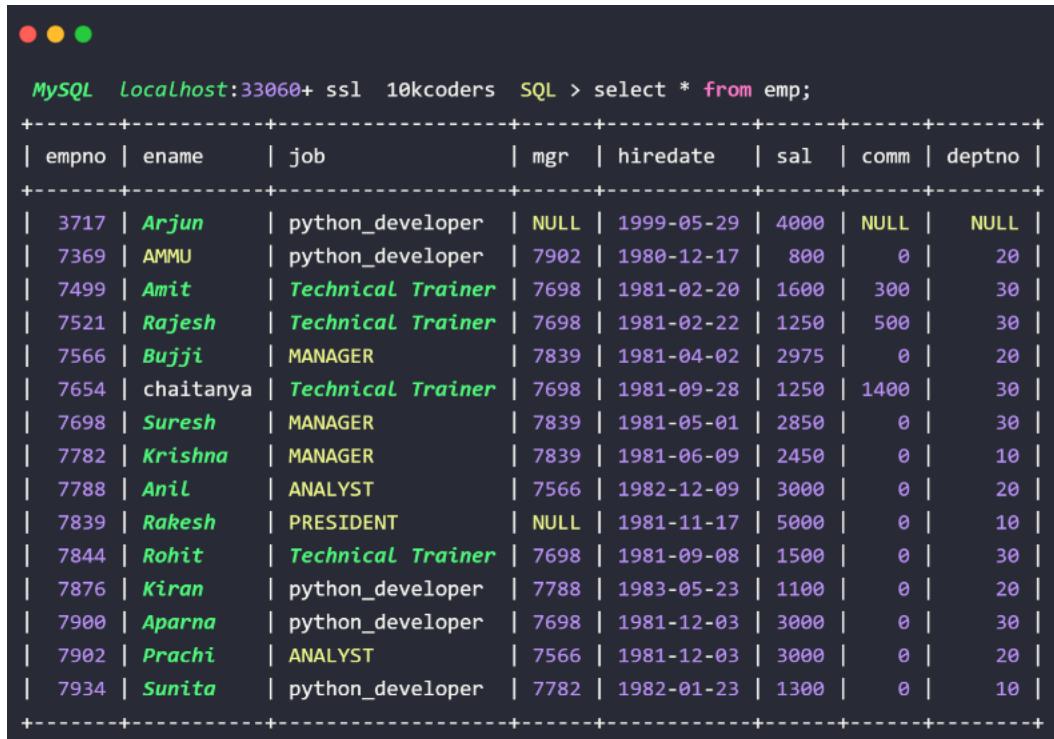
Numeric/Mathematical Functions

Function	Description	Example
ABS(number)	Absolute value	SELECT ABS(-10) FROM dual; → 10
CEIL(number)	Smallest integer \geq number	SELECT CEIL(2.3) FROM dual; → 3
FLOOR(number)	Largest integer \leq 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.

Right path for a Bright Career.

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;
```

```
+-----+  
| Highest_Salary |  
+-----+  
|      5000 |  
+-----+  
1 row in set (0.0021 sec)
```

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

3. MIN() – Lowest Salary

```
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

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

Returns the average salary of all employees.

5. COUNT() – Number of Employees

```
+-----+  
| Total_Employees |  
+-----+  
|          15 |  
+-----+  
1 row in set (0.0026 sec)
```

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

```
0 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

```
2 rows in set (0.0002 sec)
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)

```
2 rows in set (0.0422 sec)
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.

```
3 rows in set (0.0001 sec) Right path for a Bright Career
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  |
+-----+-----+-----+
3 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]
)
```

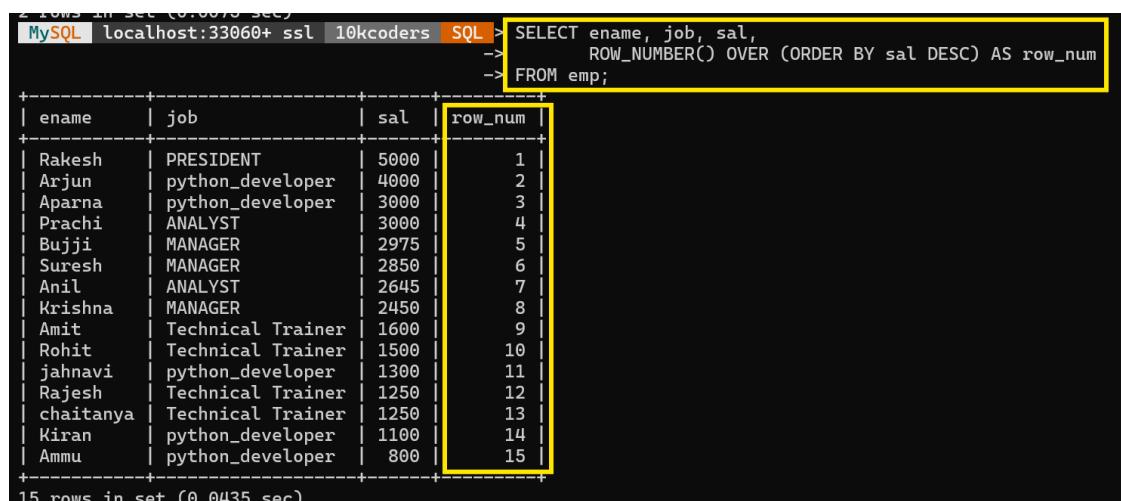
PARTITION BY → divides the result set into groups (like GROUP BY, but doesn’t collapse).

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	<input checked="" type="checkbox"/> Required	Salary rank
ROW_NUMBER()	Optional	<input checked="" type="checkbox"/> Required	Unique row ID
LAG/LEAD	Optional	<input checked="" type="checkbox"/> Required	Prev/Next row values
NTILE()	Optional	<input checked="" type="checkbox"/> 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
Ajrun	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()

Right path for a Bright Career.

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

```
15 rows in set (0.0143 sec)
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.

10kcoders
Right path for a Bright Career.

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.

Right path for a Bright Career.

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 |
+-----+-----+-----+-----+
15 rows in set (0.0062 sec)
```

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 (0.0220 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

Right path for a Bright Career.

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.

```
3 rows in set (0.1934 sec)
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)
```