RESEARCH INNOVATION DISCOVERY (RID BHARAT)
A place for coder, developer & Researcher
RUN BY TWKSAA WELFARE FOUNDATION & ISO: 9001:2015

Reg.No: 048884
Foundation Day
05-09-2024

MYSQL
Shorts Notes

Er. Rajesh Prasad(B.E, M.E)
Founder: TWF & RID Org.

- **RID ORGANIZATION** यानि **Research, Innovation and Discovery** संस्था जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इस RID संस्था** की स्थपना किया हूँ|
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

त्वक्सा MYSQL के इस ई-पुस्तक में आप MYSQL से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। मुझे आशा है कि इस ई-पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी

"In this E-Book of TWKSAA HTML you will learn all the basic concepts related to HTML. I hope after reading this E-Book your knowledge will be improve and you will get more interest to know more thing about computer Science".

RID हमें क्यों करना चाहिए ?

(Research)

अनुसंधान हमें क्यों करना चाहिए ?

Why should we do research?

1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge)
2. समस्याओं का समाधान(To Solving problems)
3. सामाजिक प्रगति (To Social progress)
4. विकास को बढ़ावा देने(To promote development)
5. तकनीकी और व्यापार में उन्नति(To advances in technology & business)
6. देश विज्ञान और प्रौद्योगिकी के विकास(To develop the country's science & technology)

(Innovation)

नवीनीकरण हमें क्यों करना चाहिए ?

Why should we do Innovation?

1. प्रगति के लिए(To progress)
2. परिवर्तन के लिए(For change)
3. उत्पादन में सुधार(To Improvement in production)
4. समाज को लाभ(To Benefit to society)
5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition)
6. देश विज्ञान और प्रौद्योगिकी के विकास। (To develop the country's science & technology)

(Discovery)

खोज हमें क्यों करना चाहिए?

Why should we do Discovery?

1. नए ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. अविष्कारों की खोज(To Discovery of inventions)
3. समस्याओं का समाधान(To Solving problems)
4. ज्ञान के विकास में योगदान(Contribution to development of knowledge)
5. समाज के उन्नति के लिए (for progress of society)
6. देश विज्ञान और तकनीक के विकास (To develop the country's science & technology)

❖ **DATA :**
- Collection of facts or information.Can be structured (organized) or unstructured (raw).
- Examples: Text, Numbers, Images, Audio, Video, Maps.

❖ **DATA STRUCTURE (डेटा संरचना)**
- Way to organize & store data in memory. Supports operations: Insert, Delete, Search, Sort, Update. **Use:** Efficient data handling in software & algorithms.

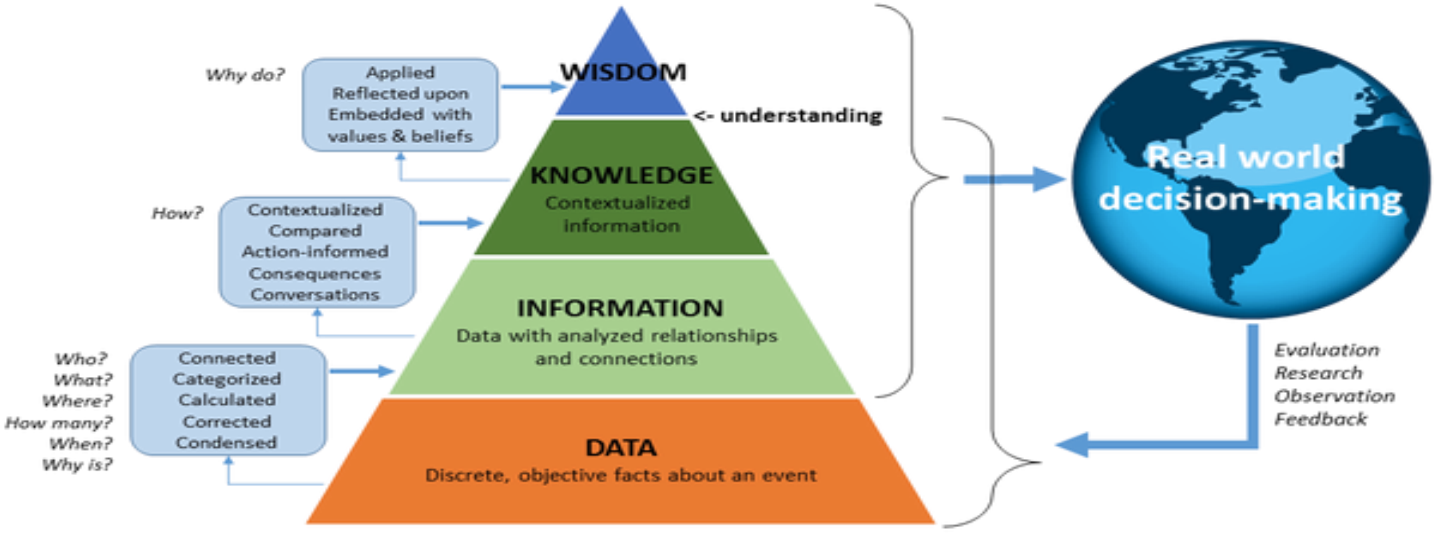
❖ **DATABASE (डेटाबेस)**
- Organized collection of data. Allows storage, retrieval & manipulation.
- **Examples:** Student Records, Banking, Social Media.

❖ **DBMS:** Software to store, organize, manage & retrieve data from a database.

❖ **Popular DBMS Examples**
- MySQL (1995, Oracle),Oracle DB (1979, Oracle),MS SQL Server (1989, Microsoft)
- PostgreSQL (1996), MongoDB (2009),SQLite (2000),IBM Db2 (1983),MariaDB (2009)
- Amazon DynamoDB (2012),Cassandra (2008)

❖ Define the Data, information, knowledge and wisdom.

1. **Data:** Data is a collection of facts, statistics, or information.
   - **Example:** Text data, Numeric data, Image data, Audio data, Video data, Geospatial data.
2. **Information:** Processed and organized data that provides context and meaning.
   - **Example:** "6 apples in a basket" or "Average marks: 6.3 inches."
3. **Knowledge:** Understanding and awareness derived from interpreting information and recognizing patterns.
   - **Example:** Knowing that "6 apples in a basket" suggests a specific quantity, or understanding the implications of "Average rainfall: 5.8 inches" for a region's climate.
4. **Wisdom**: Applying knowledge judiciously and making sound decisions based on experience and insight.
   - **Example:** Choosing to plant drought-resistant crops based on the knowledge of regional rainfall patterns and agricultural practices, demonstrating wisdom in decision-making



Relational Model (RDBMS)

- Proposed by **E.F. Codd (1970)**, it organizes data into **tables (relations)** of **rows (tuples)** and **columns (attributes)**. Implemented in **RDBMS** like MySQL, Oracle, SQL Server, PostgreSQL.

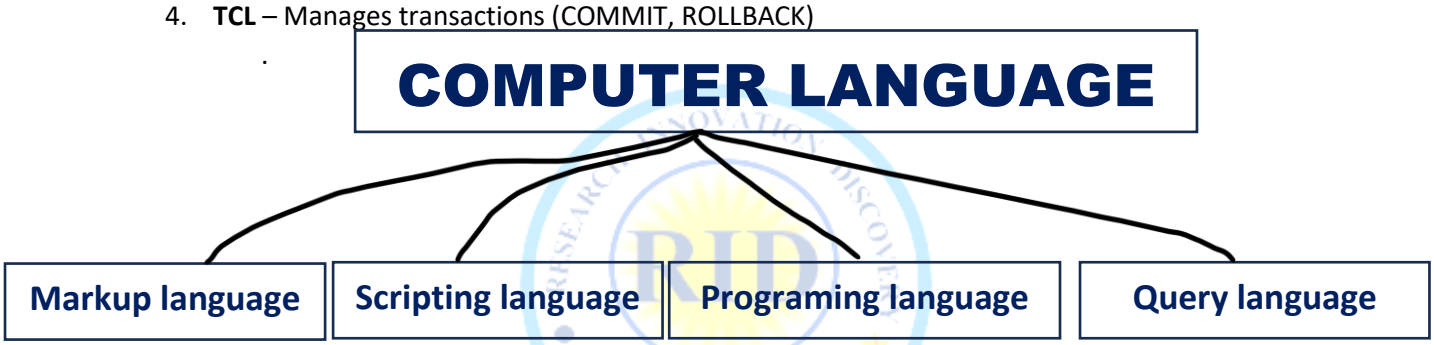
❖ **DATABASE LANGUAGE**

- **Database Language** = Language to interact with databases. **SQL** is most common
- **Examples**:
  1. SQL – Relational DBs
  2. NoSQL Queries – MongoDB (BSON), Cassandra (CQL), CouchDB (JS)
  3. GraphQL – API queries
  4. Cypher – Graph DBs (Neo4j)
  5. PL/SQL – Oracle extension of SQL
  6. T-SQL – Microsoft SQL Server extension

Types of DBMS Languages

1. **DDL** – Defines structure (CREATE, ALTER, DROP)**DQL** – Retrieves data (SELECT)
2. **DML** – Manipulates data (INSERT, UPDATE, DELETE)
3. **DCL** – Controls access (GRANT, REVOKE)
4. **TCL** – Manages transactions (COMMIT, ROLLBACK)

.



1. Markup Languages

- **Purpose:** Used to structure and present data, primarily for displaying information to humans.
- **Example:** HTML (HyperText Markup Language)
  - **Example:** <h1>This is a heading</h1> This is a paragraph of text.

2. Scripting Languages

- Automate tasks, interact with other software, and add dynamic behavior to web pages.
- **Example:** JavaScript
  - **How it works:** Used to make web pages interactive (e.g., responding to user clicks, animations, fetching data from a server).
  - **Example:** console.log("100")

3. Programming Languages : Create complex software applications, systems, and programs.

- **Example:** Python, Java, C++
  - **How it works:** Used to develop a wide range of applications, from games and mobile apps to operating systems & scientific simulations. **Ex (Python):** A=10 Print("value of a=",A)

4. Query Languages: - To retrieve specific data from databases. **Ex:** SQL (Structured Query Language)

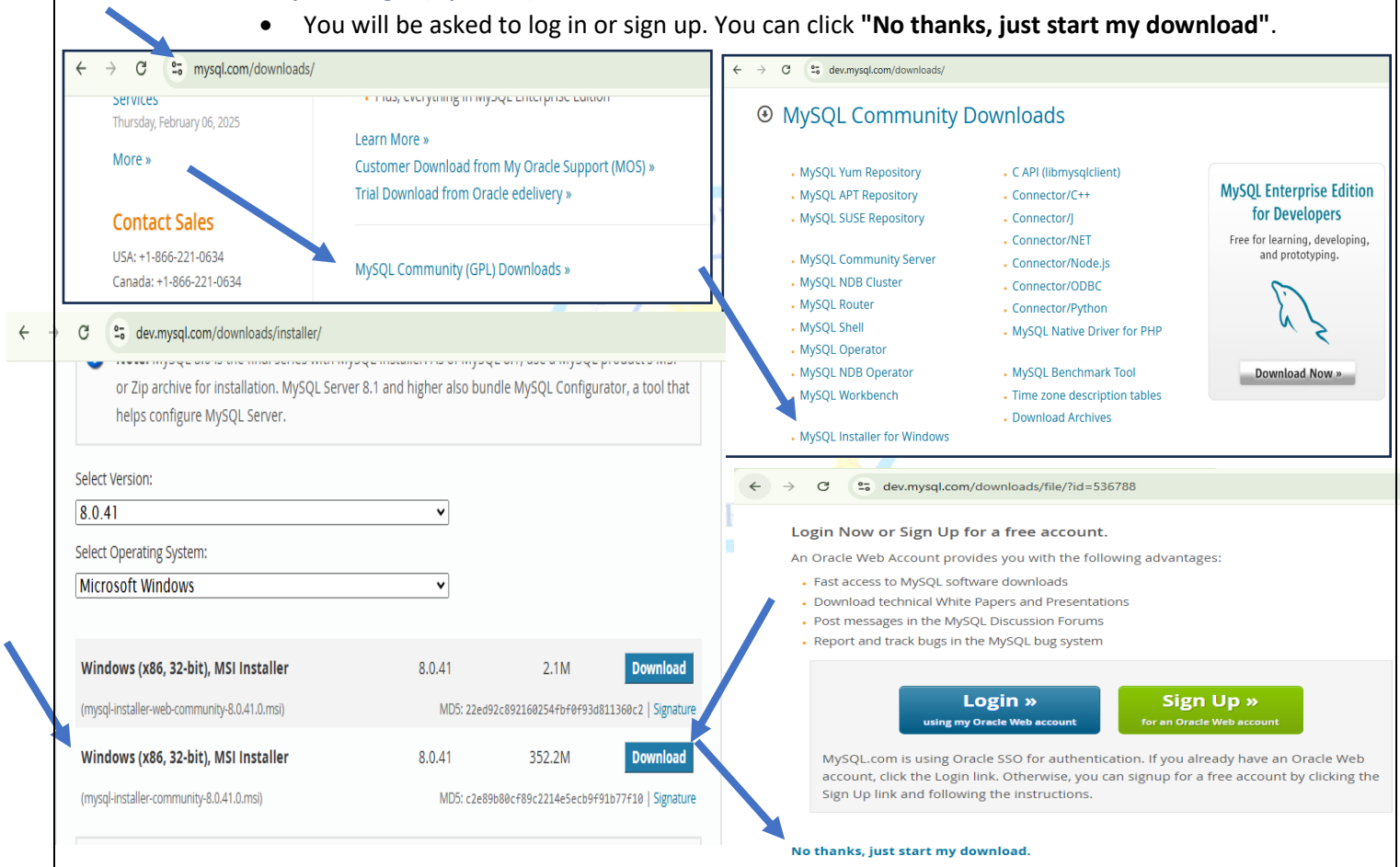
- **How it works:** Used to interact with databases (like MySQL, PostgreSQL) to retrieve, insert, update, and delete data.
- **Example:** SELECT * FROM customers WHERE city = 'New York';

Query Language: - A computer language used to **retrieve, manage, and manipulate data** in a database.

HOW TO DOWNLOAD MYSQL

Step 1: Download MySQL Installer

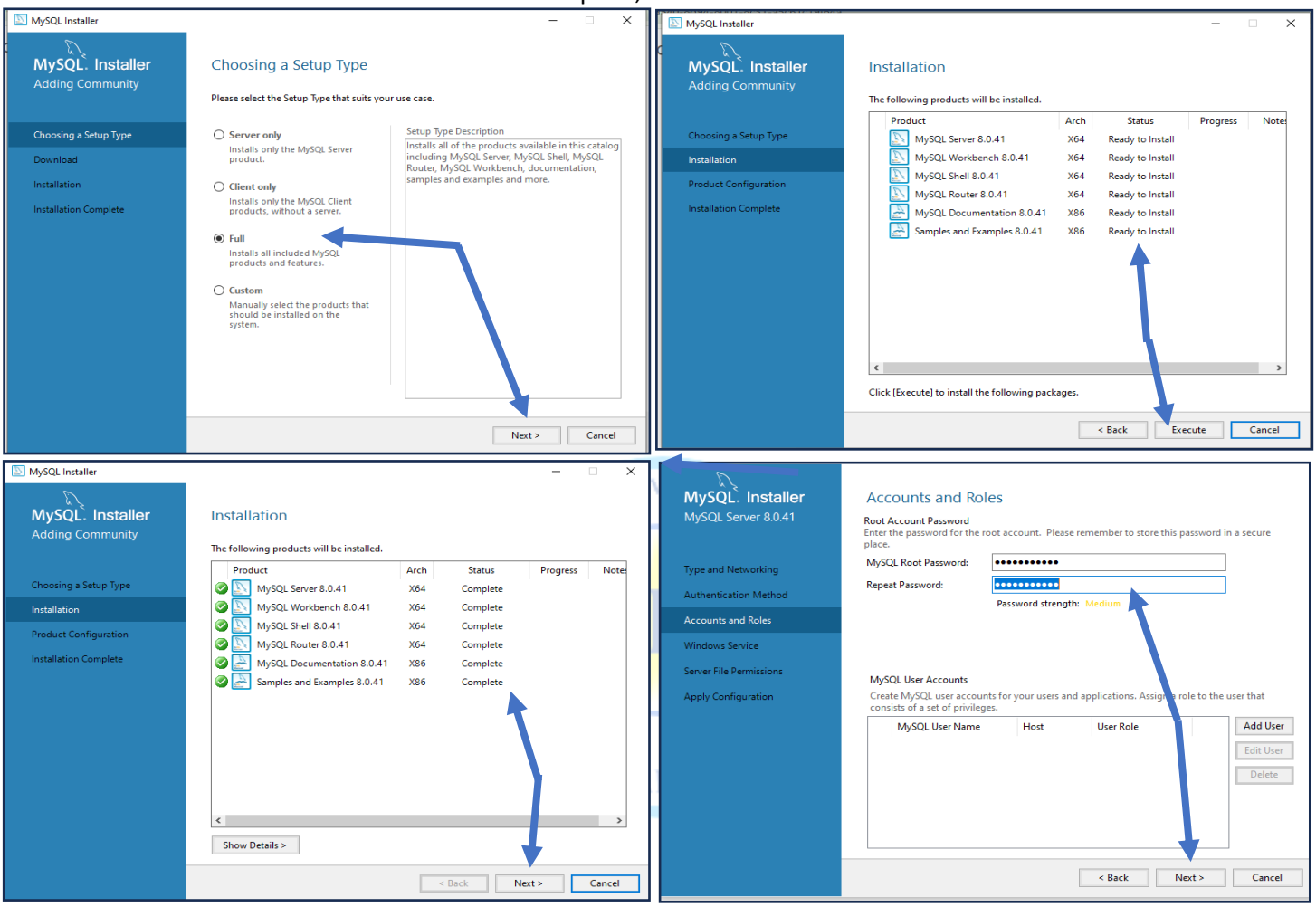
1. **Go to MySQL Official Website:**
   - Open your browser and visit: https://dev.mysql.com/downloads/.
2. **Select MySQL Installer for Windows:**
   - Click on **"MySQL Community (GPL) Downloads"**.
   - Under **"MySQL Community Downloads"**, select **MySQL Installer**.
3. **Choose the Installer Version:**
   - **If you have internet access:** Download the **smaller "web" installer** (about 2.4 MB).
   - **If you want the full package:** Download the **larger "offline" installer** (about 400 MB).
   - Click **Download**.
4. **Skip the Login (Optional):**
   - You will be asked to log in or sign up. You can click **"No thanks, just start my download"**.



Step 2: Install MySQL

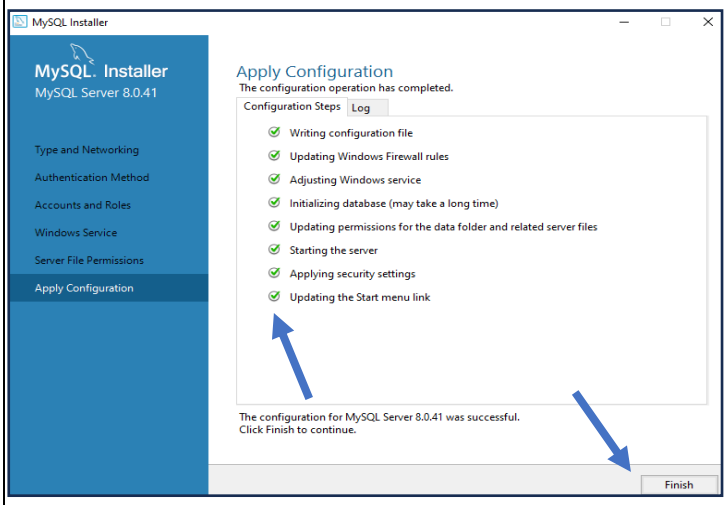
5. **Run the Installer:**
   - Locate the downloaded file (usually in the "Downloads" folder).
   - **Double-click** the installer (mysql-installer-web-community.exe or mysql-installer-community.exe).
6. **Select the Installation Type:**
   - **Developer Default** – Recommended, installs MySQL Server, Workbench, Shell, and other tools.
   - **Custom** – Allows manual selection of components.
   - Choose **Developer Default** and click **Next**.

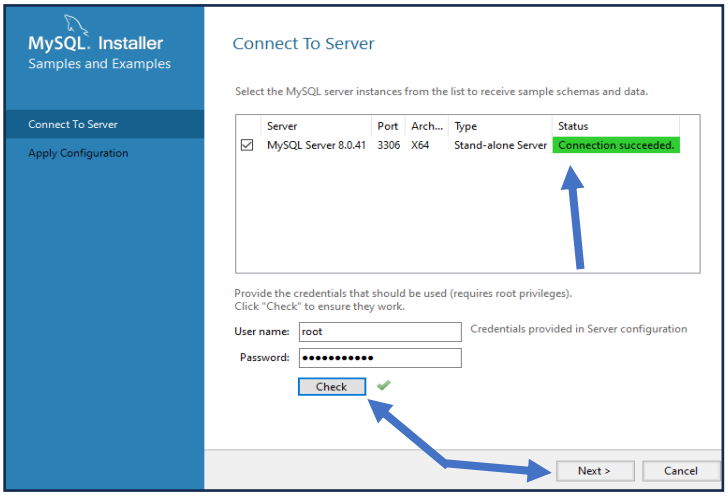
7. **Check for Requirements:**
   - The installer will check for missing dependencies (e.g., Microsoft Visual C++).
   - If anything is missing, install it as prompted.
8. **Start Installation:**
   - Click **Execute** to begin installing MySQL and Workbench.
   - Wait for the installation to complete, then click **Next**.

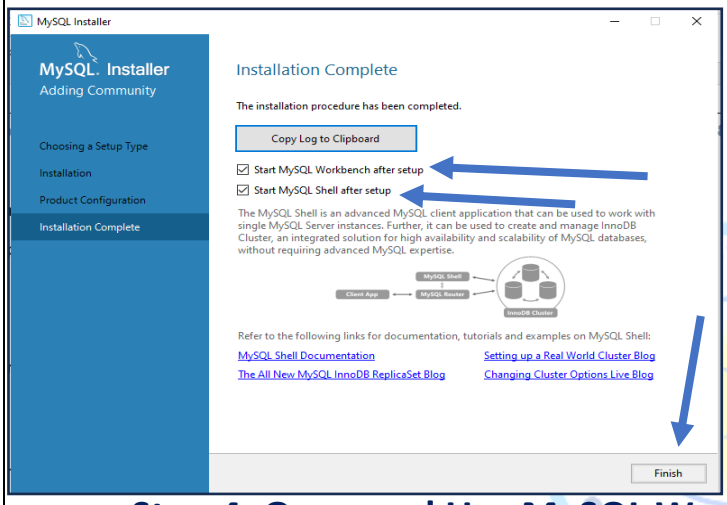


Step 3: Configure MySQL Server

9. **Select Server Type:**
   - Choose **Standalone MySQL Server** and click **Next**.
10. **Configure Authentication:**
    - Select **Use Strong Password Encryption** (recommended).
    - Set a **root password** (remember it!).
    - Click **Next**.
11. **Create a MySQL User (Optional):**
    - You can create an additional user or use the root account.
    - Click **Next**.
12. **Configure Windows Service:**
- Keep the default settings and click **Next**.
13. **Apply Configuration:**
    - Click **Execute** to apply settings.
    - Click **Finish**.

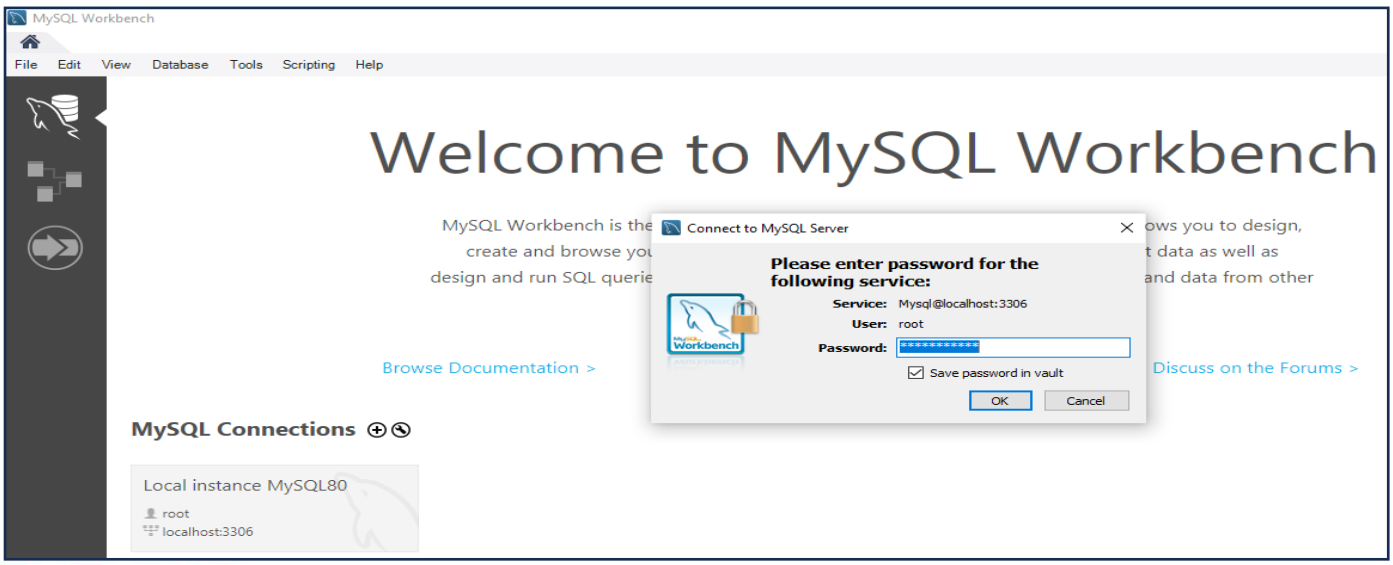


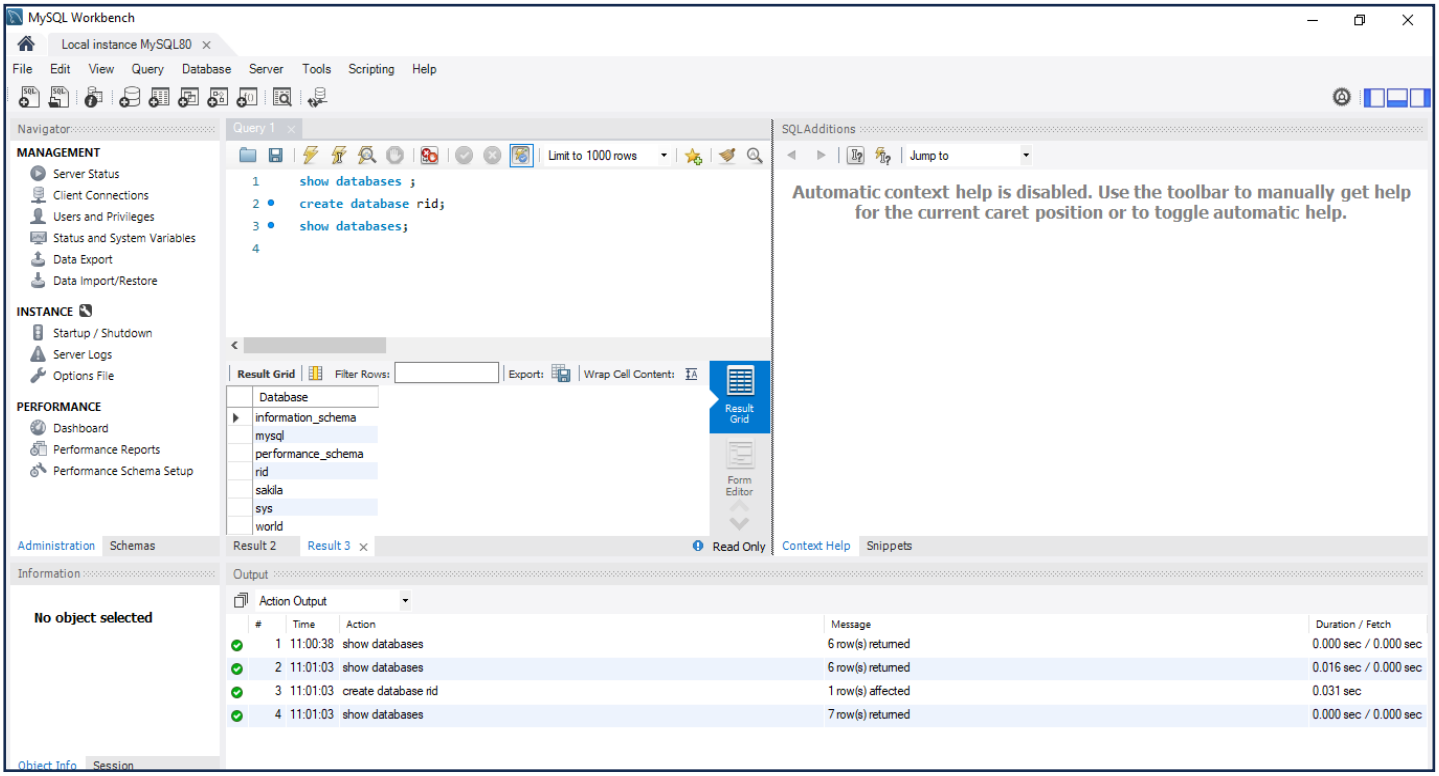




Step 4: Open and Use MySQL Workbench

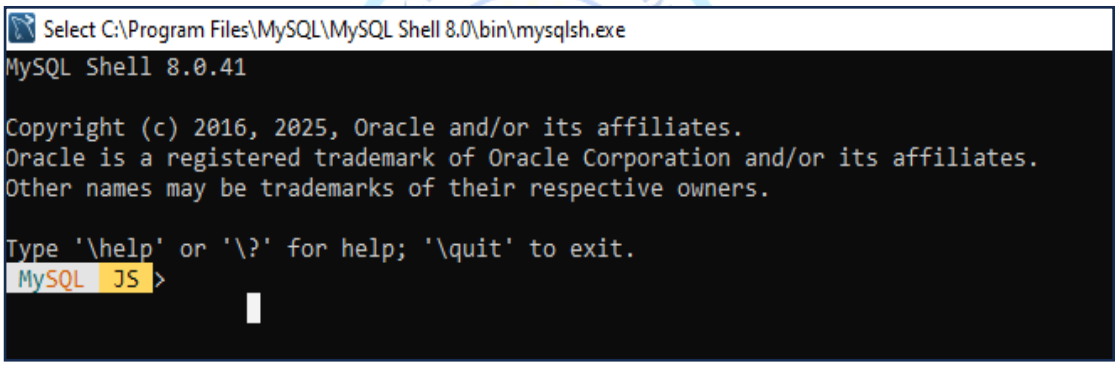
14. Launch MySQL Workbench:
    - Once installation is complete, open MySQL Workbench from the Start menu.
15. Connect to MySQL Server:
    - Click the + icon to create a new connection.
    - Enter localhost as the hostname and root as the username.
    - Enter your root password and click OK.
16. Start Using MySQL Workbench!
    - You can now create databases, run SQL queries, and manage MySQL.





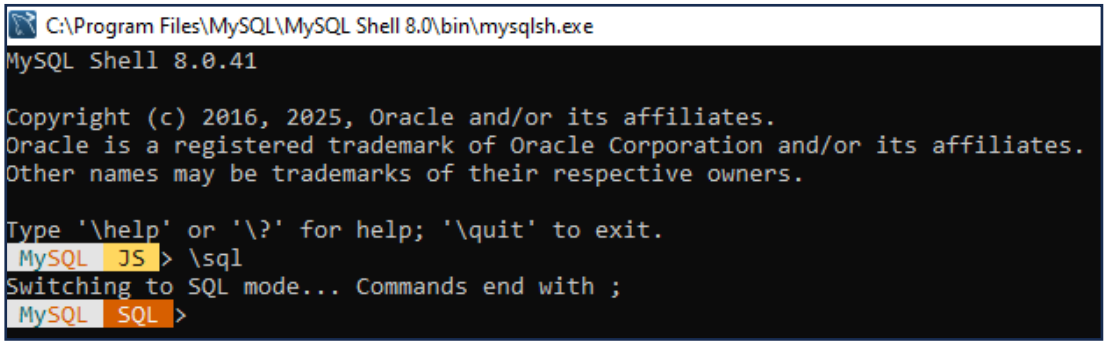
HOW TO CREATE THE DATA BASE IN MYSQL SHELL

Step-1: open MySQL shell



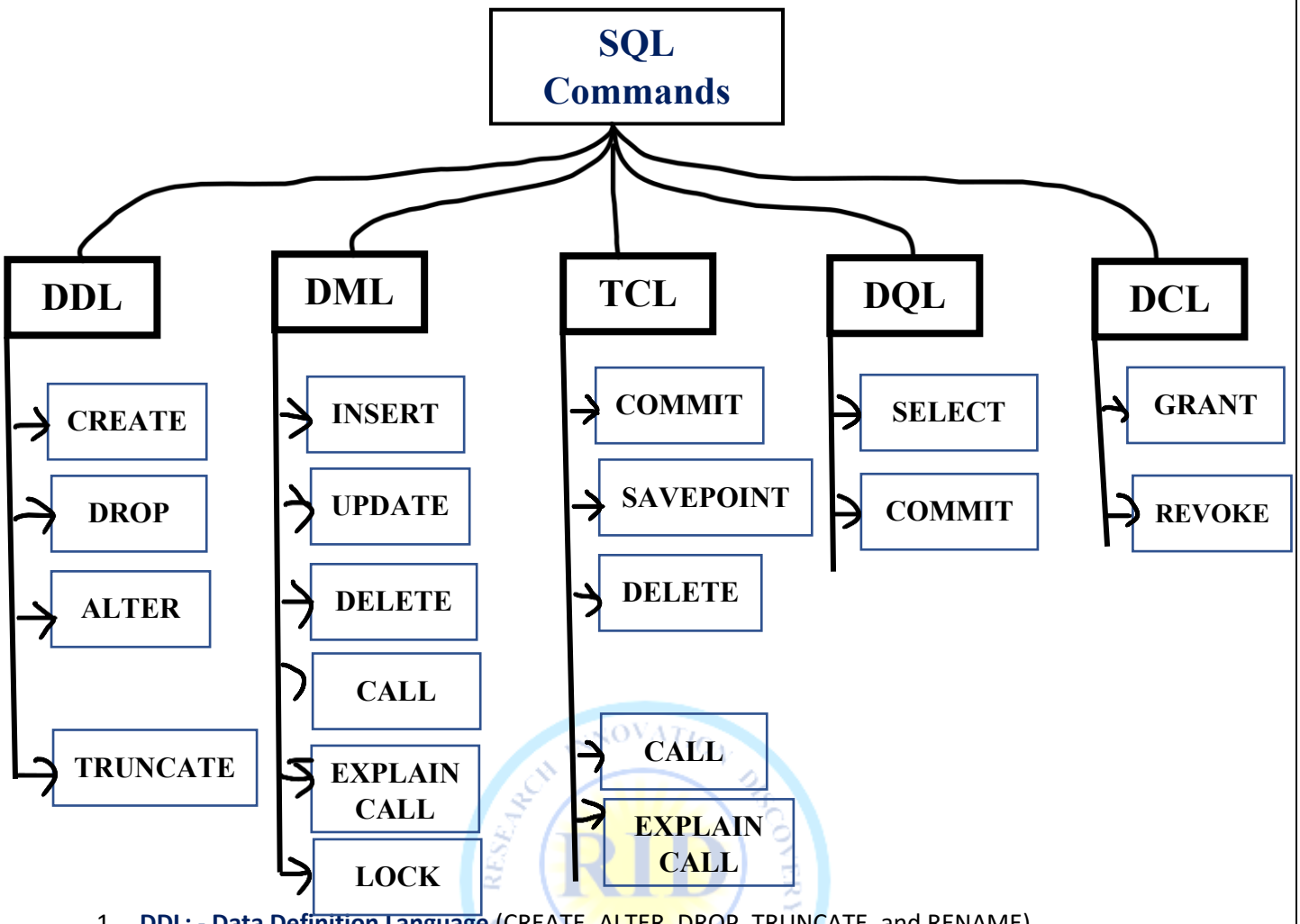
Step 1: Switch to SQL Mode if MySQL JS>

- By default, MySQL Shell may start in JavaScript (JS) mode. To use SQL mode, type:
- **MySQL JS > \sql** (Enter)
- Press Enter. Now you can run SQL queries.



Step 2: Connect to the MySQL Server

- If you haven't connected yet, use:



1. **DDL: - Data Definition Language** (CREATE, ALTER, DROP, TRUNCATE, and RENAME)
2. **DML: - Data Manipulation Language** (INSERT, UPDATE, DELETE, and MERGE)
3. **TCL: - Transaction Control Language** (OMMIT, ROLLBACK, SET TRANSACTION and SAVEPOINT)
4. **DQL: - Data Query Language (**SELECT, COMMIT**)**
5. DCL:- **Data Control Language** (GRANT, REVOKE)

1. DDL (Data Definition Language) – Defines and modifies database structures.

- **CREATE** – Creates a new database object (table, view, etc.).

➢ **Example:** CREATE TABLE students (id INT, name VARCHAR (50));

- **ALTER** – Modifies an existing database object.

➢ **Example:** ALTER TABLE students ADD age INT;**DROP** – Deletes a database object permanently.

➢ **Example:** DROP TABLE students;

- **TRUNCATE** – Removes all records from a table but keeps its structure.

➢ TRUNCATE TABLE students; **RENAME** – Changes the name of a database object.

➢ **Example:** RENAME TABLE students TO learners;

2. DML (Data Manipulation Language) – Handles data manipulation.

- **INSERT** – Adds new records to a table.

➢ **Example:** INSERT INTO students (id, name) VALUES (1, 'John');

- **UPDATE** – Modifies existing records.

➢ **Example:** UPDATE students SET name = 'Mike' WHERE id = 1; **DELETE** – Removes records from a table.

➢ **Example:** DELETE FROM students WHERE id = 1; **MERGE** – Merges data from two tables.

➢ MERGE INTO students USING new_students ON (students.id = new_students.id)
WHEN MATCHED THEN UPDATE SET students.name = new_students.name

SQL QUERY

SQL Query Basics: - Case-insensitive → Keywords in **UPPERCASE**, names in **lowercase**

Database Commands

1. **CREATE DATABASE** – Make new database → **Example:>>** CREATE DATABASE riddb;
2. **SHOW DATABASES** – List all databases → **Example:>>** SHOW DATABASES;
3. **USE** – Select a database → **Example:>>**USE riddb;
4. **SELECT DATABASE ()** – Check current database → **Example>>** SELECT DATABASE();
5. **DROP DATABASE** – Delete a database → **Example>>** DROP DATABASE riddb;
6. **SHOW DATABASES LIKE** – Check if DB exists → **Example>>** SHOW DATABASES LIKE 'riddb';
7. **SHOW TABLE STATUS** – Info about tables in DB → **Example>>** SHOW TABLE STATUS FROM riddb;
8. **SHOW CREATE DATABASE** – See DB creation SQL → **Example>>**SHOW CREATE DATABASE riddb;
9. **Show Users** – List DB users → **Example>>** SELECT User, Host FROM mysql.user;

Table Commands

1. **CREATE TABLE** – Create a new table
   Example >> CREATE TABLE student (id INT, name VARCHAR(50));
   Syntax: CREATE TABLE table_name (
   column1 datatype(size) constraints,
   column2 datatype(size) constraints,
   column3 datatype(size) constraints,
   ...);
   → **Example:** CREATE TABLE student (
   id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   age INT,
   marks DECIMAL(5,2)
   );
2. **SHOW TABLES** – List all tables in the database → **Example >>** SHOW TABLES;
3. **DESCRIBE / DESC** – Show table structure →**Example >>** DESC student;
4. **SHOW TABLES LIKE** – Check if a table exists → **Example >>** SHOW TABLES LIKE 'student';
5. **ALTER TABLE** – used for Add/modify/drop columns
   - → **Add a column** → **Example >>** ALTER TABLE student ADD roll_no INT;
   - → **Add at the beginning (first column):** → **Example>>** ALTER TABLE student ADD pincode INT FIRST;
   - → **Add after a specific column (e.g., after name):** → **Example:** ALTER TABLE student ADD Id_No INT AFTER name;
   - → **Modify a column** (e.g., change data type or size) → **Example:** ALTER TABLE student MODIFY age VARCHAR(3);
   - → **Rename the column name** →**Example>>**ALTER TABLE student RENAME COLUMN age TO stu_age;
   - → **RENAME TABLE (alternative syntax in some DBs like SQL Server)** → **Example>>** EXEC sp_rename 'student', 'learners';
   - → **Change datatype + name+size** → **Example>> ALTER TABLE student CHANGE age stu_age VARCHAR(6);**
   - → **Drop a column**(Delete the column) → **Example>>** ALTER TABLE student DROP COLUMN age;
6. **RENAME TABLE** – Rename a table → **Example >>** RENAME TABLE student TO learners;
7. **DROP TABLE** – Delete a table with its data → **Example >>** DROP TABLE student;
8. **TRUNCATE TABLE** – Delete all rows but keep structure →**Example >>** TRUNCATE TABLE student;
9. **COPY (CREATE TABLE AS SELECT)** – Copy table with data → **Example >>**CREATE TABLE newtb AS SELECT * FROM student;
10. **COUNT** – Count rows in a table →Example >> SELECT COUNT(*) FROM student;
11. **PRIMARY KEY / FOREIGN KEY** – Add constraints → **Example >>**ALTER TABLE student ADD PRIMARY KEY (id);
12. ALTER TABLE orders ADD FOREIGN KEY (id) REFERENCES student(id);
13. **DROP PRIMARY KEY / FOREIGN KEY** – Remove constraints →**Example >>**ALTER TABLE student DROP PRIMARY KEY;
    → **Example>>** ALTER TABLE orders DROP FOREIGN KEY kyname;
14. **DROP CONSTRAINT** (not only primary/foreign key, but also check/unique) →**Ex>>** ALTER TABLE student DROP CONSTRAINT constraint_name;
15. **ADD UNIQUE constraint** →**Example>>**ALTER TABLE student ADD UNIQUE (roll_no);
16. **DROP CHECK / UNIQUE constraint** Example: ALTER TABLE student DROP CONSTRAINT chk_age;
17. **AUTO_INCREMENT (MySQL) / IDENTITY (SQL Server)** To alter a column into auto incrementing:
    → Example: ALTER TABLE student MODIFY id INT AUTO_INCREMENT;
18. **ADD CHECK constraint** **Example:** ALTER TABLE student ADD CONSTRAINT chk_age CHECK (age >= 18);

Data Types in MySQL

	Type	Description	Example
1.	INT	Integer (whole number)	age INT
2.	TINYINT	Small integer	status TINYINT
3.	SMALLINT	Small integer	quantity SMALLINT
4.	BIGINT	Large integer	population BIGINT
5.	DECIMAL	Exact numeric value with decimal	price DECIMAL(10, 2)
6.	FLOAT	Floating-point number	weight FLOAT
7.	DOUBLE	Double precision floating-point number	distance DOUBLE
8.	CHAR	Fixed-length string	gender CHAR(1)
9.	VARCHAR	Variable-length string	first_name VARCHAR(50)
10.	TEXT	Long text string	description TEXT
11.	BLOB	Binary large object for binary data	image BLOB
12.	DATE	Date (YYYY-MM-DD)	birthdate DATE
13.	DATETIME	Date and time (YYYY-MM-DD HH:MM:SS)	created_at DATETIME
14.	TIMESTAMP	Timestamp (YYYY-MM-DD HH:MM:SS)	last_updated TIMESTAMP
15.	ENUM	Predefined list of valuesstatus	ENUM('active', 'inactive')
16.	SET	Multiple values from a list of options	colors SET('red', 'green', 'blue')

1. Numeric Data Types

- **INT (INTEGER)**: Whole numbers.
- **TINYINT**: Small integers.
- **SMALLINT**: Small integers.
- **MEDIUMINT**: Medium integers.
- **BIGINT**: Large integers.
- **DECIMAL (NUMERIC)**: Exact numeric values with specified precision and scale.
- **FLOAT**: Floating-point numbers.
- **DOUBLE**: Double-precision floating-point numbers.

2. **String Data Types**

- **CHAR(M):** Fixed-length string.
- **VARCHAR(M):** Variable-length string.
- **TEXT:** Long text data.
- **TINYTEXT:** Short text data.
- **MEDIUMTEXT:** Medium-length text data.
- **LONGTEXT:** Very long text data.
- **BLOB:** Binary large object (for binary data
- **TINYBLOB:** Very small binary data.
- **MEDIUMBLOB:** Medium-sized binary data.
- **LONGBLOB:** Very large binary data.

Column Descriptions in a MySQL Table Structure

Column	Description
Field	The name of the column in the table.
Type	The data type of the column (e.g., `VARCHAR(30)`, `INT`, `FLOAT`).
Null	Indicates whether the column can have `NULL` values (`YES` or `NO`).
Key	Specifies if the column is part of a key (`PRI` for Primary Key, `UNI` for Unique Key, `MUL` for Foreign Key).
Default	The default value assigned to the column if no value is provided during insertion.
Extra	Any additional information (`AUTO_INCREMENT`, `GENERATED COLUMN`, etc.).
Collation	Specifies the character set and collation used for `VARCHAR` or `TEXT` columns (e.g., `utf8mb4_general_ci`).
Privileges	Defines the permissions on the column (e.g., `SELECT`, `INSERT`, `UPDATE`, etc.).
Comment	Stores additional information or description for the column (optional).

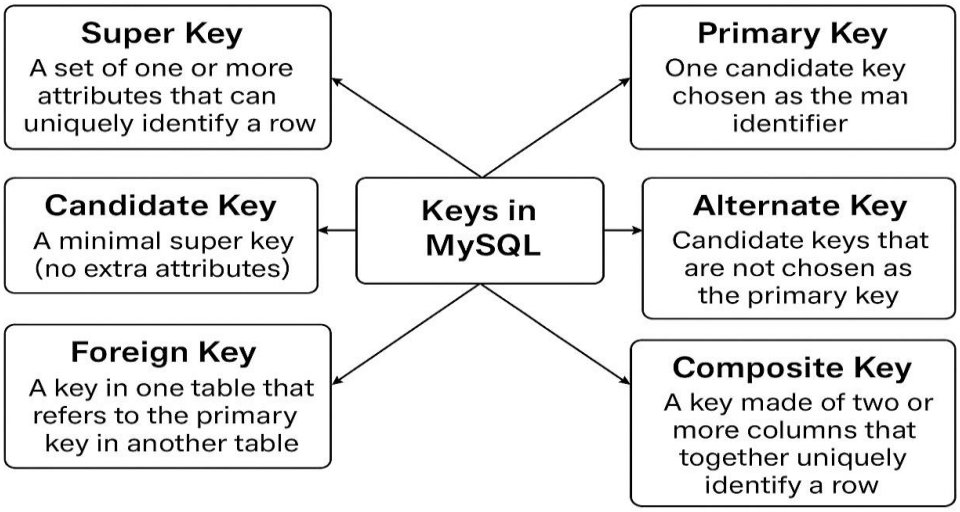
Keys in MySQL

A **key** is an attribute (or set of attributes) that uniquely identifies a record in a table.

Purpose: To uniquely identify rows. To create relationships between tables.

Types of Keys

1. **Super Key :-** A set of one or more attributes that can uniquely identify a row. Can have extra attributes.
   - **Example:** {Roll_No}, {Aadhar_No}, {Email_id}, {Roll_No, Name}
2. **Candidate Key** :- A minimal super key (no extra attributes). Uniquely identifies each row.
   - **Example:** {Roll_No}, {Aadhar_No}, {Email_id}
3. **Primary Key**:- One candidate key chosen as the main identifier. Must be **unique** and **not null**.
   - Only one primary key per table.
   - **Example:** {Roll_No}
4. **Alternate Key** :- Candidate keys that are **not chosen** as the primary key.
   - **Example:** If Roll_No is primary → Aadhar_No, Email_id become alternate keys.
5. **Foreign Key :-** A key in one table that refers to the **primary key** in another table. Maintains **referential integrity** between tables.
   - Example: Department_id in Student table → refers to Department_id in Department table.
6. **Composite Key** A key made of **two or more columns** that together uniquely identify a row.
   - **Example:** {Roll_No, Department_id}



1. Add a Primary Key to an Existing Table

- **Syntax:** ALTER TABLE table_name ADD PRIMARY KEY (column_name);
- **Example:** ALTER TABLE students ADD PRIMARY KEY (student_id);

2. Add a Foreign Key to a Table

- **Syntax:** ALTER TABLE table_name ADD CONSTRAINT fk_name FOREIGN KEY (column_name) REFERENCES other_table (column_name);
- **Example:** ALTER TABLE students ADD CONSTRAINT fk_course_id FOREIGN KEY (course_id) REFERENCES courses(course_id);

3.Add a Composite Primary Key (Multiple Columns)

- A Composite Primary Key is a primary key using multiple columns.
- **Syntax:** ALTER TABLE table_name ADD PRIMARY KEY (column1, column2);
- **Example:** ALTER TABLE enrollments ADD PRIMARY KEY (student_id, course_id);

Note: If table raj already has a primary key, trying to add another one will result in this error.

4.Add an Auto Increment Constraint

- The AUTO_INCREMENT attribute allows a column to automatically generate unique values.
- It is generally used on int data type value and
- **Syntax:** ALTER TABLE table_name MODIFY column_name INT AUTO_INCREMENT;
- **Example:** ALTER TABLE students MODIFY student_id INT AUTO_INCREMENT;

5.Drop a Primary Key

- **Syntax:** ALTER TABLE table_name DROP PRIMARY KEY;
- **Example:** ALTER TABLE students DROP PRIMARY KEY;

6.Drop a Foreign Key

- **Syntax:** ALTER TABLE table_name DROP FOREIGN KEY fk_name;
- **Example:** ALTER TABLE students DROP FOREIGN KEY fk_course_id;

7.Show Table Status

- To get detailed information about a table (e.g., number of rows, storage engine).
- **Syntax:** SHOW TABLE STATUS LIKE 'table_name';
- **Example:** SHOW TABLE STATUS LIKE 'students';

8.Create Table Like (Copy Table Structure Only)

- Creates a table with the same structure (no data) as an existing table.
- **Syntax:** CREATE TABLE new_table_name LIKE existing_table_name;
- **Example:** CREATE TABLE students_copy LIKE students;

9.Add a Unique Constraint

- Ensures that all values in a column are unique.
- **Syntax:-** ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column_name);
- **Example:-** ALTER TABLE students ADD CONSTRAINT unique_student_email UNIQUE (email);

10Drop a Constraint (Primary Key, Foreign Key, Unique Key)

- Removes a constraint (primary key, foreign key, or unique key) from a table.
  - **Syntax:** ALTER TABLE table_name DROP CONSTRAINT constraint_name;
  - **Example (to drop a unique constraint):**
  - ALTER TABLE students DROP CONSTRAINT unique_student_email;

11.Create a View: - Creates a view, which is a virtual table based on a query.

Syntax:

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

Example:

- CREATE VIEW student_view AS
  SELECT student_id, first_name, last_name
  FROM students
  WHERE age > 18;

12 Drop a View: - Removes an existing view from the database.

- **Syntax:** DROP VIEW view_name;
- **Example:** DROP VIEW student_view;

13.Show Triggers on a Table:- Displays triggers attached to a table.

- **Syntax:** SHOW TRIGGERS LIKE 'table_name';**Example:** SHOW TRIGGERS LIKE 'students';

14.Check Auto-Increment Information

- To check the current auto-increment value for a table.
  - **Syntax:** SHOW TABLE STATUS LIKE 'table_name';
  - **Example:** SHOW TABLE STATUS LIKE 'students'

INSERT DATA IN TABLE

Method	Use Case
1) INSERT INTO ... VALUES	Insert a single row
2) INSERT INTO ... VALUES (Multiple Rows)	Insert multiple rows efficiently
3) INSERT INTO ... SET	Insert a single row with SET syntax
4) INSERT INTO ... SELECT	Copy data from another table

1. INSERT INTO ... VALUES (Single Row) Used to insert a single record into a table.

- **Syntax: INSERT INTO** table_name (column1, column2, column3, ...)
  VALUES (value1, value2, value3, ...);
- **Example: -** mysql> INSERT INTO raj(Name, age,marks,Gender)
  VALUES("Sangam Kuamr", 15, 90.05, "Male");

2. INSERT INTO ... VALUES (Multiple Rows) Used to insert multiple records in a single query.

- **Syntax:** INSERT INTO table_name (column1, column2, column3, ...)
  VALUES (value1a, value2a, value3a, ...),
  (value1b, value2b, value3b, ...),
  (value1c, value2c, value3c, ...);
- **Example:** mysql> INSERT INTO raj(Name, age,marks,Gender)
  VALUES ("Satyam Kumar", 20, 99.1, "Male"),
  ("Shushil Kumar", 18, 98, "Male"),
  ("Roshni Kumari", 17, 93, "Female");

3.INSERT INTO ... SET :- Inserts a **single record** using SET instead of VALUES. This method is useful for inserting values without **specifying** column names in parentheses.

Syntax: INSERT INTO table_name SET
column1 = value1,
column2 = value2,
column3 = value3;

Example: INSERT INTO students SET id = 5, name = 'Sangam Kumar', age = 24;

4.INSERT INTO ... SELECT (Copy data from another table)

- This method is used to copy data from one table to another.
- Create two table with name attributes then apply

- **Use Case:** Copy data from one table (raj2) to another (raj) based on a condition.

a). Create Tables (raj and raj2)

```
CREATE TABLE raj (
  name VARCHAR(50),
  id INT, age INT );
```

```
CREATE TABLE raj2 (
  Name VARCHAR(50),
  id INT,
  age INT);
```

b). Insert Sample Data into raj2

```
INSERT INTO raj2 (Name, id, age) VALUES
('Rahul Sharma', 101, 25),
('Priya Verma', 102, 30),
('Amit Kumar', 103, 28);
```

c). Copy Data from raj2 to raj (Only for age < 26)

```
INSERT INTO raj (id, name, age)
SELECT id, Name, age FROM raj2 WHERE age < 26;
```

d). Verify Data in raj: SELECT * FROM raj;

Logical Operations, Operators, and Control Statements in MySQL

1. Logical Operators in MySQL Logical operators are used to perform boolean logic in SQL queries.

Operator	Description	Example
AND	Returns TRUE if both conditions are true.	SELECT * FROM students WHERE age > 18 AND city = 'Delhi';
OR	Returns TRUE if at least one condition is true.	SELECT * FROM students WHERE age > 18 OR city = 'Delhi';
NOT	Negates a condition (returns the opposite boolean value).	SELECT * FROM students WHERE NOT city = 'Delhi';
XOR	Returns TRUE if one condition is true but not both.	SELECT * FROM students WHERE age > 18 XOR city = 'Delhi';

2. Operators in MySQL MySQL provides various operators for different operations

a). Arithmetic Operators

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**

b). Comparison Operators:

Operator	Description	Example
=	Equal to	SELECT * FROM students WHERE age = 18;
!= or <>	Not equal to	SELECT * FROM students WHERE age != 18;
>	Greater than	SELECT * FROM students WHERE age > 18;
<	Less than	SELECT * FROM students WHERE age < 18;
>=	Greater than or equal to	SELECT * FROM students WHERE age >= 18;
<=	Less than or equal to	SELECT * FROM students WHERE age <= 18;
BETWEEN	Checks if value is within a range	SELECT * FROM students WHERE age BETWEEN 10 AND 20;
IN	Checks if value exists in a list	SELECT * FROM students WHERE city IN ('Delhi', 'Mumbai');
LIKE	Pattern matching using wildcards	SELECT * FROM students WHERE name LIKE 'A%';
IS NULL	Checks for NULL values	SELECT * FROM students WHERE marks IS NULL;

3. Control Statements in MySQL Control statements allow conditional execution and looping in stored procedures and functions.

a). Conditional Control Statements

Statement	Description	Example
IF	Executes a block of code if a condition is true.	IF age >= 18 THEN SELECT 'Adult'; ELSE SELECT 'Minor'; END IF;
CASE	Evaluates conditions and executes matching block.	SELECT CASE WHEN age >= 18 THEN 'Adult' ELSE 'Minor' END AS category FROM students;

b). Looping Control Statements

Statement	Description	Example
LOOP	Repeats a block of code until an exit condition is met.	LOOP SELECT 'Looping'; END LOOP;
WHILE	Repeats a block of code while a condition is true.	WHILE counter < 10 DO SET counter = counter + 1; END WHILE;

c). Iteration Control Statement

Statement	Description	Example
LEAVE	Exits a loop.	LEAVE loop_label;
ITERATE	Skips the current iteration and moves to the next.	ITERATE loop_label;

4. Constraints in SQL

- Constraints ensure the integrity and validity of data in a database.

Constraint	Description	Example
PRIMARY KEY	Ensures uniqueness of a column.	CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50));
FOREIGN KEY	Establishes a relationship between tables.	CREATE TABLE marks (student_id INT, FOREIGN KEY (student_id) REFERENCES students(id));
UNIQUE	Ensures column values are unique.	CREATE TABLE users (email VARCHAR(100) UNIQUE);
NOT NULL	Prevents null values in a column.	CREATE TABLE employees (id INT NOT NULL, name VARCHAR(50));
CHECK	Ensures column values meet a condition.	CREATE TABLE persons (age INT CHECK (age >= 18));
DEFAULT	Assigns a default value if no value is provided.	CREATE TABLE orders (status VARCHAR(10) DEFAULT 'Pending');

5. Joins in SQL Joins combine rows from multiple tables based on a related column.

Join Type	Description	Example
INNER JOIN	Returns matching records from both tables.	SELECT students.name, marks.score FROM students INNER JOIN marks ON students.id = marks.student_id;
LEFT JOIN	Returns all records from the left table and matching records from the right.	SELECT students.name, marks.score FROM students LEFT JOIN marks ON students.id = marks.student_id;
RIGHT JOIN	Returns all records from the right table and matching records from the left.	SELECT students.name, marks.score FROM students RIGHT JOIN marks ON students.id = marks.student_id;
FULL JOIN	Returns all records when there is a match in either table.	SELECT students.name, marks.score FROM students FULL JOIN marks ON students.id = marks.student_id;
SELF JOIN	Joins a table with itself.	SELECT A.name, B.name FROM employees A, employees B WHERE A.manager_id = B.id;
CROSS JOIN	Returns the Cartesian product of two tables.	SELECT * FROM products CROSS JOIN categories;

6. Common SQL Functions: - SQL functions perform calculations on data.

a). Aggerate function

Function	Description	Example
COUNT ()	Counts the number of rows.	SELECT COUNT (*) FROM students;
SUM ()	Returns the sum of values.	SELECT SUM (salary) FROM employees;
AVG ()	Returns the average value.	SELECT AVG (marks) FROM students;
MIN ()	Returns the minimum value.	SELECT MIN (salary) FROM employees;
MAX ()	Returns the maximum value.	SELECT MAX (salary) FROM employees;

b). String Functions

Function	Description	Example
LENGTH()	Returns the length of a string.	SELECT LENGTH('Hello'); → **5**
UPPER()	Converts text to uppercase.	SELECT UPPER('hello'); → **HELLO**
LOWER()	Converts text to lowercase.	SELECT LOWER('HELLO'); → **hello**
SUBSTRING()	Extracts part of a string.	SELECT SUBSTRING('HelloWorld', 1, 5); → **Hello**

C). Date Functions

Function	Description	Example
NOW()	Returns the current timestamp.	SELECT NOW();
CURDATE()	Returns the current date.	SELECT CURDATE();
CURTIME()	Returns the current time.	SELECT CURTIME();
DATE_FORMAT()	Formats a date.	SELECT DATE_FORMAT(NOW(), '%Y-%m-%d');

8.Transactions in SQL: - Transactions ensure data integrity.

Command	Description	Example
START TRANSACTION	Begins a transaction.	START TRANSACTION;
COMMIT	Saves changes permanently.	COMMIT;
ROLLBACK	Reverts changes.	ROLLBACK;
SAVEPOINT	Creates a rollback point.	SAVEPOINT sp1;

9. Wildcards in SQL (Used with LIKE)

- **Wildcards help filter data in pattern matching.**

Wildcard	Description	Example
%	Represents any sequence of characters.	SELECT * FROM students WHERE name LIKE 'A%';
_	Represents a single character.	SELECT * FROM students WHERE name LIKE 'A_';

10. Subqueries in SQL: - A subquery is a query inside another query.

Type	Description	Example
Scalar Subquery	Returns a single value.	SELECT name FROM students WHERE id = (SELECT MAX(id) FROM students);
Multiple-row Subquery	Returns multiple values.	SELECT name FROM students WHERE id IN (SELECT id FROM marks WHERE score > 80);
Correlated Subquery	Depends on the outer query.	SELECT name FROM students S WHERE EXISTS (SELECT * FROM marks M WHERE S.id = M.student_id);

SELECT QUERY

1. Basic SELECT Query → Retrieves all or specific columns from a table.

→ **Syntax:** SELECT column1, column2 FROM table_name **Example:** SELECT name, age FROM students;

2. SELECT All Columns (*) → Fetches all columns from a table.

→ **Syntax:** SELECT * FROM table_name; **Example:** SELECT * FROM student;

3. SELECT with WHERE Clause → Filters data based on a condition.

→ **Syntax:** SELECT column1 FROM table_name WHERE condition; **Ex:** SELECT name FROM student WHERE age > 18;

4. SELECT with ORDER BY (Sorting) → Sorts results in ascending (ASC) or descending (DESC) order.

→ **Syntax:** SELECT column1 FROM table_name ORDER BY column1 ASC|DESC;

→ **Example:** SELECT name, age FROM student ORDER BY age DESC;

5. SELECT with DISTINCT (Unique Values) Retrieves unique values from a column.

→ **Syntax:** SELECT DISTINCT column1 FROM table_name; → **Ex:** SELECT DISTINCT city FROM student;

6. SELECT with LIMIT (Fetch Specific Rows)→ Restricts the number of rows in the result.

→ **Syntax:** SELECT column1 FROM table_name LIMIT number; **Example:** SELECT * FROM student LIMIT 5;

7. SELECT with LIKE (Pattern Matching) Searches for a pattern in a column using % and _.

→ **Syntax:** SELECT column1 FROM table_name WHERE column1 LIKE 'pattern';

→ **Example:** SELECT name FROM student WHERE name LIKE 'A%'; n Finds names starting with "A")

8. SELECT with IN (Multiple Values) Filters rows that match values in a list.

→ **Syntax:** SELECT column1 FROM table_name WHERE column1 IN (value1, value2);

→ **Example:** SELECT name FROM student WHERE city IN ('Delhi', 'Mumbai');

9. SELECT with BETWEEN (Range) Retrieves values between two numbers or dates.

→ **Syntax:** SELECT column1 FROM table_name WHERE column1 BETWEEN value1 AND value2;

→ **Example:** SELECT name FROM student WHERE age BETWEEN 18 AND 25;

10. SELECT with GROUP BY (Grouping Data) Groups rows with the same values and allows aggregate functions.

→ **Syntax:** SELECT column1, COUNT(*) FROM table_name GROUP BY column1;

→ **Example:** SELECT city, COUNT(*) FROM student GROUP BY city;

11. SELECT with HAVING (Filter After Grouping) Filters grouped results using HAVING.

→ **Syntax:** SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING COUNT(*) > value;

→ **Example:** SELECT city, COUNT(*) FROM students GROUP BY city HAVING COUNT(*) > 5;

→**Example:** CREATE TABLE student (id INT PRIMARY KEY AUTO_INCREMENT,name VARCHAR(100),city VARCHAR(100));

Step 2: Insert sample student data

INSERT INTO student (name, city) VALUES('Rahul Sharma', 'Delhi'),('Amit Kumar', 'Mumbai'),('Priya Verma', 'Delhi'),('Neha Singh', 'Kolkata'),('Vikash Patel', 'Mumbai'),('Suresh Raina', 'Kolkata'),('Ravi Kumar', 'Delhi');

Step 3: Execute the GROUP BY query

→SELECT city, COUNT(*) AS student_count FROM student GROUP BY city;

12. SELECT with JOIN (Combining Tables) Retrieves data from multiple tables.

→**Syntax:** SELECT table1.column1, table2.column2 FROM table1 JOIN table2 ON table1.common_column = table2.common_column;

→**Example:** SELECT students.name, courses.course_name FROM students JOIN courses ON students.course_id = courses.id;

- **UPDATE:** Used to change data in a table.
- **SET:** Tells **which column** you want to change and the **new value**.
- **WHERE:** Tells **which row(s)** should be updated. Without WHERE, all rows will be updated.

UPDATE QUERIES

1. UPDATE (Change Data in a Row) →Modify column values for a specific row.

- **Syntax:** UPDATE table_name SET column1 = value1 WHERE condition;
- **Example:** UPDATE students SET age = 20 WHERE id = 1; changes age to 20 for student with ID 1

2. UPDATE Multiple Columns →Change more than one column at once.

- **Syntax:** UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
- **Example:** UPDATE students SET age = 21, city = 'Delhi' WHERE id = 2;

3. UPDATE All Rows (No WHERE)→ Update all rows (⚠ be careful).

- **Syntax:** UPDATE table_name SET column1 = value1;
- **Example:** UPDATE students SET city = 'Mumbai'; sets city as Mumbai for all students

4. UPDATE with IN (Multiple Rows) →Update specific rows using a list.

- **Syntax:** UPDATE table_name SET column1 = value1 WHERE column2 IN (value1, value2);
- **Example:** UPDATE students SET grade = 'A' WHERE id IN (3, 5, 7);

5. UPDATE with LIMIT →Update limited number of rows.

- **Syntax:** UPDATE table_name SET column1 = value1 WHERE condition LIMIT number;
- **Example:** UPDATE students SET age = 22 WHERE city = 'Delhi' LIMIT 5;

6. UPDATE with CASE (Conditional Update) →Update values based on conditions.

Syntax:
```
UPDATE table_name SET column1 = CASE
    WHEN condition1 THEN value1
    WHEN condition2 THEN value2
    ELSE value3
END;
```

Example:
```
UPDATE students
SET grade = CASE
   WHEN marks >= 90 THEN 'A'
   WHEN marks >= 75 THEN 'B'
   ELSE 'C'
END;
```

Difference Between UPDATE and MODIFY in MySQL

Feature	UPDATE	MODIFY (ALTER TABLE MODIFY)
Purpose	Changes existing **data** in a table.	Changes the **structure** of a column in a table.
Used For	Updating values in rows (records).	Modifying data type, size, or attributes of a column.
Affects	Row values (data inside columns).	Column definition (not the data).
Command Type	DML (Data Manipulation Language).	DDL (Data Definition Language).
WHERE Clause	Needed to update specific rows.	Not applicable (affects column structure).
Rollback Possible?	Yes, can be rolled back using ROLLBACK (if within a transaction).	No, changes are permanent.
Example Usage	UPDATE students SET age = 20 WHERE id = 1;	ALTER TABLE students MODIFY age INT(3);

DELETE QUERY

1. DELETE (Remove Specific Rows): - Deletes specific rows from a table based on a condition.

- **Syntax:** DELETE FROM table_name WHERE condition;
- **Example:** DELETE FROM students WHERE age < 18;
- **Note: -** (Deletes all students younger than 18.) Without WHERE, all rows will be deleted!

2. DELETE All Rows (Without Condition) Deletes all rows but keeps the table structure.

- **Syntax:** DELETE FROM table_name;**Example:** DELETE FROM students;
- **Note:** (Removes all records from the students table.) Unlike TRUNCATE, this can be rolled back if inside a transaction.

3. DELETE with LIMIT (Delete Specific Number of Rows) Deletes only a limited number of rows.

- **Syntax:** DELETE FROM table_name WHERE condition LIMIT number;
- **Example:** DELETE FROM students WHERE age > 25 LIMIT 5;
- **Note:** (Deletes only 5 students older than 25.)

4. DELETE with ORDER BY (Delete Oldest/Newest Data First) Deletes rows in a specific order.

- **Syntax:** DELETE FROM table_name WHERE condition ORDER BY column ASC|DESC LIMIT number;
- **Example:** DELETE FROM students ORDER BY id DESC LIMIT 3;
- **Note:** (Deletes the last 3 inserted records.)

5. DELETE with JOIN (Remove Data from Multiple Tables):- Deletes rows based on another table's condition.

- **Syntax:** DELETE table1 FROM table1
  JOIN table2 ON table1.common_column = table2.common_column
  WHERE condition;
- **Example:**
  DELETE students FROM students
  JOIN courses ON students.course_id = courses.id
  WHERE courses.name = 'Math';
- **Note:** (Deletes students enrolled in the "Math" course.)

6. TRUNCATE (Delete All Rows and Reset IDs) Removes all data from a table and resets auto-increment.

- **Syntax:** TRUNCATE TABLE table_name; **Example:** TRUNCATE TABLE students;
- **Note:** Deletes all students and resets the ID counter. Faster than DELETE FROM table_name; Cannot be rolled back!

7. DROP TABLE (Completely Remove a Table) Deletes the entire table along with its structure.

- **Syntax:** DROP TABLE table_name;**Example:** DROP TABLE students;

Note: Completely removes the student's table. This action is permanent and cannot be undone!

Query	Purpose	Can Be Rolled Back?
DELETE FROM table WHERE condition;	Deletes specific rows	Yes
DELETE FROM table;	Deletes all rows but keeps structure	Yes
DELETE FROM table ORDER BY column LIMIT X;	Deletes a limited number of rows	Yes
DELETE FROM table USING JOIN;	Deletes based on another table	Yes
TRUNCATE TABLE table;	Deletes all rows and resets auto-increment	No
DROP TABLE table;	Deletes the table permanently	No
DROP DATABASE db;	Deletes the entire database	No

Example-1

1. **Create database MySQL>**Create database rid;
2. **Use database MySQL>**Use rid;
3. **Show database MySQL>**Select databases();
4. **Create table**

MySQL>CREATE TABLE student (
sid INT PRIMARY KEY,
name VARCHAR(30),
Branch VARCHAR(30),
city VARCHAR(30),
marks INT, ,age INT);

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| sid    | int         | YES  |     | NULL    |       |
| name   | varchar(30) | YES  |     | NULL    |       |
| Branch | varchar(30) | YES  |     | NULL    |       |
| city   | varchar(30) | YES  |     | NULL    |       |
| marks  | int         | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

5. **Insert Student Records in table**

mysql>INSERT INTO student (sid, name, Branch, city, marks, age) VALUES
(1, 'Rahul Sharma', 'Computer Science', 'Delhi', 85, 20),
(2, 'Amit Kumar', 'Mechanical', 'Mumbai', 78, 21),
(3, 'Priya Verma', 'Electrical', 'Kolkata', 92, 19),
(4, 'Neha Singh', 'Electronics', 'Chennai', 88, 22),
(5, 'Vikash Patel', 'Civil', 'Bangalore', 75, 23),
(6, 'Ravi Ranjan', 'IT', 'Hyderabad', 80, 20),
(7, 'Suresh Raina', 'Computer Science', 'Ahmedabad', 95, 21),
(8, 'Anjali Pandey', 'Mechanical', 'Pune', 72, 22),
(9, 'Suman Tiwari', 'Electrical', 'Jaipur', 89, 19),
(10, 'Pankaj Mishra', 'Electronics', 'Lucknow', 77, 20);

```
mysql> select *from student;
+-----+--------------+-----------------+-----------+-------+-----+
| sid | name         | Branch          | city      | marks | age |
+-----+--------------+-----------------+-----------+-------+-----+
|   1 | Rahul Sharma | Computer Science | Delhi     |    85 |  20 |
|   2 | Amit Kumar   | Mechanical      | Mumbai    |    78 |  21 |
|   3 | Priya Verma  | Electrical      | Kolkata   |    92 |  19 |
|   4 | Neha Singh   | Electronics     | Chennai   |    88 |  22 |
|   5 | Vikash Patel | Civil           | Bangalore |    75 |  23 |
|   6 | Ravi Ranjan  | IT              | Hyderabad |    80 |  20 |
|   7 | Suresh Raina | Computer Science | Ahmedabad |    95 |  21 |
|   8 | Anjali Pandey | Mechanical     | Pune      |    72 |  22 |
|   9 | Suman Tiwari | Electrical      | Jaipur    |    89 |  19 |
|  10 | Pankaj Mishra | Electronics    | Lucknow   |    77 |  20 |
|  11 | Divya Kapoor | Civil           | Chandigarh |    91 |  22 |
|  12 | Manish Gupta | IT              | Patna     |    84 |  23 |
|  13 | Arun Joshi   | Computer Science | Indore    |    76 |  19 |
|  14 | Kavita Mehta | Mechanical      | Bhopal    |    83 |  21 |
|  15 | Rohit Sen    | Electrical      | Nagpur    |    90 |  22 |
+-----+--------------+-----------------+-----------+-------+-----+
15 rows in set (0.00 sec)
```

6. **How to retrieve all data from the table**

Syntax: select *from table_name; Example: mysql>select *from student;

7. **How to print the specific column**

Syntax: select column1, column2,table_name; **Example:** mysql> select sid, name,age from student;

8. **How to select sid as student_id from table**

Syntax: select sid as student id from table_name; **Example:** mysql> select sid as student_id from student;

9. **How retrieve data from table according to the condition**

Syntax: select column-1, column-2 from table name where condition;

Example: mysql> select *from student where sid=6;

```
mysql> select *from student where sid=6;
+------+-------------+--------+-----------+-------+------+
| sid  | name        | Branch | city      | marks | age  |
+------+-------------+--------+-----------+-------+------+
|    6 | Ravi Ranjan | IT     | Hyderabad |    80 |   20 |
+------+-------------+--------+-----------+-------+------+
1 row in set (0.00 sec)
```

10. **Select student detail where name is Priya Verma**

Example: mysql> select *from student where name="priya verma";

```
mysql> select *from student where name="priya verma";
+------+-------------+------------+---------+-------+------+
| sid  | name        | Branch     | city    | marks | age  |
+------+-------------+------------+---------+-------+------+
|    3 | Priya Verma | Electrical | Kolkata |    92 |   19 |
+------+-------------+------------+---------+-------+------+
1 row in set (0.00 sec)
```

11.Select student detail where branch is cse

Example: mysql> select *from student where name="priya verma";

12.mysql> select *from student where branch="Computer Science";

```
mysql> select *from student where branch="Computer Science";
+------+--------------+------------------+-----------+-------+------+
| sid  | name         | Branch           | city      | marks | age  |
+------+--------------+------------------+-----------+-------+------+
|    1 | Rahul Sharma | Computer Science | Delhi     |    85 |   20 |
|    7 | Suresh Raina | Computer Science | Ahmedabad |    95 |   21 |
|   13 | Arun Joshi   | Computer Science | Indore    |    76 |   19 |
+------+--------------+------------------+-----------+-------+------+
3 rows in set (0.00 sec)
```

13.Select the student details where branch is cse or IT;

mysql> select *from student where branch="Computer Science" or branch= “IT”;

```
mysql> select *from student where branch="Computer Science" or branch= "IT" ;
+------+--------------+------------------+-----------+-------+------+
| sid  | name         | Branch           | city      | marks | age  |
+------+--------------+------------------+-----------+-------+------+
|    1 | Rahul Sharma | Computer Science | Delhi     |    85 |   20 |
|    6 | Ravi Ranjan  | IT               | Hyderabad |    80 |   20 |
|    7 | Suresh Raina | Computer Science | Ahmedabad |    95 |   21 |
|   12 | Manish Gupta | IT               | Patna     |    84 |   23 |
|   13 | Arun Joshi   | Computer Science | Indore    |    76 |   19 |
+------+--------------+------------------+-----------+-------+------+
5 rows in set (0.00 sec)
```

14.Select the student details where branch is cse and city is indore;

mysql> select *from student where branch="Computer Science" and city="indore";

```
mysql> select *from student where branch="Computer Science" and city="indore";
+------+------------+------------------+--------+-------+------+
| sid  | name       | Branch           | city   | marks | age  |
+------+------------+------------------+--------+-------+------+
|   13 | Arun Joshi | Computer Science | Indore |    76 |   19 |
+------+------------+------------------+--------+-------+------+
1 row in set (0.00 sec)
```

15.Show all details

```
mysql> select *from student;
+------+---------------+--------+-------------+----------------+------+-------+
| sid  | Name          | Branch | Address     | City           | Age  | Marks |
+------+---------------+--------+-------------+----------------+------+-------+
|  101 | Sangam        | CSE    | SSM         | Patna          |   18 |    10 |
|  102 | Shushil       | IT     | manar       | Hajipur        |   20 |    12 |
|  103 | Satyam        | ME     | Bhojpur     | ARA            |   19 |    15 |
|  101 | Rahul Kumar   | CSE    | Patna       | Patna          |   20 |    10 |
|  102 | Anjali Verma  | ECE    | Muzaffarpur | Muzaffarpur    |   18 |    12 |
|  103 | Satyam        | ME     | Bhojpur     | Ara            |   19 |    15 |
|  104 | Priya Singh   | EEE    | Gaya        | Gaya           |   21 |    18 |
|  105 | Amit Thakur   | Civil  | Bhagalpur   | Bhagalpur      |   22 |    20 |
|  106 | Sneha Kumari  | IT     | Darbhanga   | Darbhanga      |   20 |    22 |
|  107 | Vikas Jha     | ME     | Purnia      | Purnia         |   23 |    25 |
|  108 | Ravi Raj      | CSE    | Chapra      | Saran          |   19 |    30 |
|  109 | Neha Sharma   | ECE    | Samastipur  | Samastipur     |   18 |    35 |
|  110 | Manish Pandey | Civil  | Begusarai   | Begusarai      |   21 |    40 |
|  111 | Rohit Sinha   | CSE    | Siwan       | Siwan          |   22 |    45 |
|  112 | Pooja Kumari  | ECE    | Motihari    | East Champaran |   19 |    50 |
|  113 | Aditya Narayan| ME     | Sasaram     | Rohtas         |   20 |    55 |
|  114 | Kajal Mishra  | IT     | Munger      | Munger         |   21 |    60 |
|  115 | Suresh Yadav  | Civil  | Katihar     | Katihar        |   23 |    65 |
| NULL | NULL          | NULL   | NULL        | NULL           | NULL |    95 |
+------+---------------+--------+-------------+----------------+------+-------+
19 rows in set (0.00 sec)
```

16. Select student details who are from cse or ece or eee

mysql> select *from student where branch="cse" or branch="ece" or branch="me";

mysql> select name from student where branch="cse" or branch="ece" or branch="me";

mysql> select age from student where branch="cse" or branch="ece" or branch="me";

mysql> select age from student where branch in("cse", "ece");

mysql> select name, age, branch from student where branch in("cse", "ece");

```
mysql> select *from student where branch="cse" or branch="ece" or branch="me";
+------+---------------+--------+------------+----------------+------+-------+
| sid  | Name          | Branch | Address    | City           | Age  | Marks |
+------+---------------+--------+------------+----------------+------+-------+
|  101 | Sangam        | CSE    | SSM        | Patna          |   18 |    10 |
|  103 | Satyam        | ME     | Bhojpur    | ARA            |   19 |    15 |
|  101 | Rahul Kumar   | CSE    | Patna      | Patna          |   20 |    10 |
|  102 | Anjali Verma  | ECE    | Muzaffarpur| Muzaffarpur    |   18 |    12 |
|  103 | Satyam        | ME     | Bhojpur    | Ara            |   19 |    15 |
|  107 | Vikas Jha     | ME     | Purnia     | Purnia         |   23 |    25 |
|  108 | Ravi Raj      | CSE    | Chapra     | Saran          |   19 |    30 |
|  109 | Neha Sharma   | ECE    | Samastipur | Samastipur     |   18 |    35 |
|  111 | Rohit Sinha   | CSE    | Siwan      | Siwan          |   22 |    45 |
|  112 | Pooja Kumari  | ECE    | Motihari   | East Champaran |   19 |    50 |
|  113 | Aditya Narayan| ME     | Sasaram    | Rohtas         |   20 |    55 |
+------+---------------+--------+------------+----------------+------+-------+
11 rows in set (0.00 sec)
```

17. Select student details who are not from cse or ece or eee

mysql> select name, age, branch from student where branch not in("cse", "ece");

```
mysql> select name, age, branch from student where branch not in("cse", "ece");
+----------------+------+--------+
| name           | age  | branch |
+----------------+------+--------+
| Shushil        |   20 | IT     |
| Satyam         |   19 | ME     |
| Satyam         |   19 | ME     |
| Priya Singh    |   21 | EEE    |
| Amit Thakur    |   22 | Civil  |
| Sneha Kumari   |   20 | IT     |
| Vikas Jha      |   23 | ME     |
| Manish Pandey  |   21 | Civil  |
| Aditya Narayan |   20 | ME     |
| Kajal Mishra   |   21 | IT     |
| Suresh Yadav   |   23 | Civil  |
+----------------+------+--------+
11 rows in set (0.00 sec)
```

18. Show student name, sid who marks is greater than 90

mysql> select name, sid, marks from student where marks>90;

mysql> select name, sid, marks from student where marks<50;

mysql> select name, sid, marks from student where marks<50 and marks>20;

mysql> select *from student where marks<50 and marks>20;

mysql> select city from student where marks<50 and marks>20;

```
mysql> select *from student where marks<50 and marks>20;
+------+---------------+--------+------------+------------+------+-------+
| sid  | Name          | Branch | Address    | City       | Age  | Marks |
+------+---------------+--------+------------+------------+------+-------+
|  106 | Sneha Kumari  | IT     | Darbhanga  | Darbhanga  |   20 |    22 |
|  107 | Vikas Jha     | ME     | Purnia     | Purnia     |   23 |    25 |
|  108 | Ravi Raj      | CSE    | Chapra     | Saran      |   19 |    30 |
|  109 | Neha Sharma   | ECE    | Samastipur | Samastipur |   18 |    35 |
|  110 | Manish Pandey | Civil  | Begusarai  | Begusarai  |   21 |    40 |
|  111 | Rohit Sinha   | CSE    | Siwan      | Siwan      |   22 |    45 |
+------+---------------+--------+------------+------------+------+-------+
6 rows in set (0.00 sec)
```

19. How to Display unique branch name from student

Distinct →to get unique value

mysql> select distinct (branch) from student;

mysql> select distinct (name) from student;

```
mysql> select distinct (branch) from student;
+--------+
| branch |
+--------+
| CSE    |
| IT     |
| ME     |
| ECE    |
| EEE    |
| Civil  |
| NULL   |
+--------+
7 rows in set (0.01 sec)
```

20. Find the minimum marks from the students min() are used

mysql> select **min(marks)** from student;

mysql> select min(marks) from student where branch="ece";

mysql> select min(marks) name from student where branch="ece";

21. How to find the maximum marks from the table : max() are used

mysql> select max(marks) from student where branch="ece";

22. Nested Query

Display student id and marks who having maximum marks

mysql> select name, sid from student where marks=(select max(marks) from student);

Empty set (0.00 sec)

mysql> select name, sid from student where marks> (select max(marks) from student where branch="cse");

```
mysql> select name, sid from student where marks> (select max(marks) from student where branch="cse");
+----------------+------+
| name           | sid  |
+----------------+------+
| Pooja Kumari   |  112 |
| Aditya Narayan |  113 |
| Kajal Mishra   |  114 |
| Suresh Yadav   |  115 |
| NULL           | NULL |
+----------------+------+
5 rows in set (0.53 sec)
```

mysql> select name, sid from student where marks> (select min(marks) from student where branch="cse");

mysql> select name, sid,marks from student where marks> (select min(marks) from student where branch="cse");

mysql> select name, sid,marks, branch from student where marks> (select min(marks) from student where branch="cse");

mysql> select name, sid,marks, branch from student where marks> (select max(marks) from student where branch="cse");

mysql> select name, sid,marks, branch, city from student where marks> (select max(marks) from student where city="patna");

mysql> select name, sid,marks, branch, city from student where marks> (select max(marks) from student where city="gaya");

23.How to find the sum of all marks By using the sum()

mysql> select sum(marks) from student;

24.How to find the average of the marks

mysql> select avg(marks) from student;

25.How to count the number of student By using the count()

- ➔ It does not accept the null value but it will count the duplicates values
- ➔ Display the number student in a table

mysql> select count(marks) from student;

mysql> select count(name) from student;

26. Display the number of distinct location

mysql> select (count(distinct city)) from student;

mysql> select count(*) from student where age> 18 and age < 25;

mysql> select count(*) from student where age> 20 and age < 25;

mysql> select count(*) from student where age> 20 or age < 25;

AGGERATE FUNCTION

Function	Description	Example
COUNT ()	Counts the number of rows.	SELECT COUNT (*) FROM students;
SUM ()	Returns the sum of values.	SELECT SUM (salary) FROM employees;
AVG ()	Returns the average value.	SELECT AVG (marks) FROM students;
MIN ()	Returns the minimum value.	SELECT MIN (salary) FROM employees;
MAX ()	Returns the maximum value.	SELECT MAX (salary) FROM employees;

1. **Count ():** Counts the number of rows.

   Syntax : SELECT COUNT (*) FROM students

Query-1 display the number of students

mysql> select count(*) from students;

```
mysql> select count(*) from students;
+----------+
| count(*) |
+----------+
|       10 |
+----------+
1 row in set (0.00 sec)
```

- **Using COUNT with a Specific Column**
  - **mysql>** SELECT COUNT(id) FROM students;
  - **Note:** Counts the number of non-null values in the id column.
- **Using COUNT with DISTINCT (Unique Values)**
  - **mysql>** SELECT COUNT(DISTINCT age) FROM students;
  - **Note:** Counts the number of unique ages in the table.
- **Using COUNT with an Alias**
  - **mysql>** SELECT COUNT(*) AS total_students FROM students;
  - **Note:** Renames the result as total_students.
- **Using COUNT with GROUP BY**
- **mysql>** SELECT age, COUNT(*) AS student_count
  FROM students
  GROUP BY age;
- **Note:** Counts the number of students in each age group.
- **Using COUNT with HAVING Clause**
- **Mysql:** SELECT age, COUNT(*) AS total_students FROM students GROUP BY age
  HAVING COUNT(*) > 2;
- **Note:** Shows age groups where more than 2 students exist.

GROUPING

grouping is done using the GROUP BY clause, which groups rows with the same values in specified columns. It is commonly used with **aggregate functions** like SUM(), COUNT(), AVG(), MAX(), and MIN() to perform calculations on each group.

27.Display the number of student in each branch

mysql> select branch, count(*) from student group by branch;

```
mysql> select branch, count(*) from student group by branch;
+--------+----------+
| branch | count(*) |
+--------+----------+
| CSE    |        4 |
| IT     |        3 |
| ME     |        4 |
| ECE    |        3 |
| EEE    |        1 |
| Civil  |        3 |
| NULL   |        1 |
+--------+----------+
7 rows in set (0.00 sec)
```

28.Select minimum marks in each branch

mysql> select branch, min(marks) from student group by branch;

```
mysql> select branch, min(marks) from student group by branch;
+--------+------------+
| branch | min(marks) |
+--------+------------+
| CSE    |         10 |
| IT     |         12 |
| ME     |         15 |
| ECE    |         12 |
| EEE    |         18 |
| Civil  |         20 |
| NULL   |         95 |
+--------+------------+
7 rows in set (0.00 sec)
```

29. Select the student id, name, branch who got minimum marks in every Department

mysql> SELECT sid, name, branch, marks FROM student s WHERE marks = (SELECT MIN(marks) FROM student WHERE branch = s.branch);

```
mysql> SELECT sid, name, branch, marks FROM student s
    -> WHERE marks = (SELECT MIN(marks) FROM student WHERE branch = s.branch);
+------+--------------+--------+-------+
| sid  | name         | branch | marks |
+------+--------------+--------+-------+
|  101 | Sangam       | CSE    |    10 |
|  102 | Shushil      | IT     |    12 |
|  103 | Satyam       | ME     |    15 |
|  101 | Rahul Kumar  | CSE    |    10 |
|  102 | Anjali Verma | ECE    |    12 |
|  103 | Satyam       | ME     |    15 |
|  104 | Priya Singh  | EEE    |    18 |
|  105 | Amit Thakur  | Civil  |    20 |
+------+--------------+--------+-------+
8 rows in set (0.00 sec)
```

30. Select address, no of students in each location

Mysql> select address, count(*) from student group by address;

```
+--------------+------------+
| address      | count(*)   |
+--------------+------------+
| SSM          |          1 |
| manar        |          1 |
| Bhojpur      |          2 |
| Patna        |          1 |
| Muzaffarpur  |          1 |
| Gaya         |          1 |
| Bhagalpur    |          1 |
| Darbhanga    |          1 |
| Purnia       |          1 |
| Chapra       |          1 |
| Samastipur   |          1 |
| Begusarai    |          1 |
| Siwan        |          1 |
| Motihari     |          1 |
| Sasaram      |          1 |
| Munger       |          1 |
| Katihar      |          1 |
| NULL         |          1 |
+--------------+------------+
18 rows in set (0.00 sec)
```

Having clause

HAVING clause in MySQL is used to filter **groups** after applying the GROUP BY clause. It works like the WHERE clause but is used for **aggregate functions** like SUM(), COUNT(), AVG(), etc.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example:

```
SELECT branch, AVG(marks) AS avg_marks
        FROM student
        GROUP BY branch
        HAVING avg_marks > 50;
```

Key Differences Between WHERE and HAVING:

- WHERE is used **before** grouping to filter individual rows.
- HAVING is used **after** grouping to filter aggregated results.

31.Display the city name that city having more than 2 students

mysql> select city,count(*) from student group by city having count(*)>2;
Empty set (0.00 sec)
mysql> select city,count(*) from student group by city having count(*)>1;
mysql> select city,count(*) from student group by city having count(*)>=1;
mysql> select city,count(*) from student group by city having count(*)>=2;

32.Select the average marks of student based on branch

mysql> Select branch, avg(marks) from student group by branch;

```
mysql> Select branch, avg(marks) from student group by branc
+--------+-------------------+
| branch | avg(marks)        |
+--------+-------------------+
| CSE    |             23.75 |
| IT     | 31.333333333333332 |
| ME     |              27.5 |
| ECE    | 32.333333333333336 |
| EEE    |                18 |
| Civil  | 41.666666666666664 |
| NULL   |                95 |
+--------+-------------------+
7 rows in set (0.00 sec)
```

33.Update the student city name where sid=101

mysql> update student set city="Sasaram" WHERE SID=101;
Query OK, 2 rows affected (0.01 sec)
mysql> select *from student;
mysql> UPDATE STUDENT SET age=16, marks=99 where name="Sangam";
Query OK, 1 row affected (0.01 sec)
mysql> update student set sid=116, name="anu", address="ssm", branch="cse", age=8
where city="patna" and marks=95;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

34.Delete the particular record

Syntax: delete from table_name where condation
mysql> delete from student where sid=109;
Query OK, 1 row affected (0.01 sec)

35.Limit

LIMIT clause in MySQL is used to restrict the number of rows returned by a query. It is useful when fetching a specific number of records, such as in pagination or retrieving top results.

Syntax: SELECT column_name(s) FROM table_name LIMIT number_of_rows;
SELECT column_name(s) FROM table_name LIMIT offset, number_of_rows;

- **number_of_rows**: The number of records to retrieve.
- **offset**: The starting position (zero-based index).

Example:

1. **Retrieve the first 5 rows:** SELECT * FROM students LIMIT 3;
2. **Retrieve 5 rows starting from the 3rd row (offset 2):** SELECT * FROM students LIMIT 2, 5;
   mysql> select *from student limit 3;
   3 rows in set (0.00 sec)
   mysql> delete from student where branch="cse" or city="patna";
   Query OK, 18 rows affected (0.01 sec)

STRING FUNCTIONS

Function	Description	Example
1. LENGTH()	Returns the length of a string.	SELECT LENGTH('Hello'); → **5**
2. UPPER() or ucase()	Converts text to uppercase.	SELECT UPPER('hello'); → **HELLO**
3. LOWER() or lcase()	Converts text to lowercase.	SELECT LOWER('HELLO'); → **hello**
4. SUBSTRING()	Extracts part of a string.	SELECT SUBSTRING('HelloWorld', 1, 5); → **Hello**

5. DEFAULT	Sets the default value of marks to 0 (optional).	ALTER TABLE student ADD COLUMN marks INT DEFAULT 0;
6. **Concat()**	It is used for concocts two string	SELECT CONCAT(fname, ' ', lname) AS full_name FROM student;
7. concat_ws()	**concatenates multiple strings** with a specified separator.	SELECT CONCAT_WS("--", sid, fname, lname, city, age) FROM student;
8. Substring()	tracts a portion of a string based on a specified starting position and length.	SUBSTRING(string, start_position, length)
SELECT SUBSTRING(fname, 1, 3) FROM student; Extract last four digits of sid: **Example:** SELECT SUBSTRING (sid, -4) FROM student; SELECT SUBSTRING(dob, 1, 4) FROM student;		
9. REPLACE()	replaces all occurrences of a specified substring within a string with a new substring.	
syntax	REPLACE(original_string, search_string, replace_with)	
Example	SELECT REPLACE(city, 'Delhi', 'New Delhi') FROM student;	
10. reverse()	reverses the order of characters in a string.	
Syntax	REVERSE(string)	
Example	SELECT REVERSE(fname) FROM student;	

Mysql> create table student(sid int primary key auto_increment, fname varchar(30), lname varchar(30), city varchar(30), age int, dob date);

```
mysql> desc student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| fname | varchar(30) | YES  |     | NULL    |                |
| lname | varchar(30) | YES  |     | NULL    |                |
| city  | varchar(30) | YES  |     | NULL    |                |
| age   | int         | YES  |     | NULL    |                |
| dob   | date        | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
6 rows in set (0.02 sec)
```

Mysql> ALTER TABLE student ADD COLUMN marks INT DEFAULT 0;

```
mysql> desc student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| fname | varchar(30) | YES  |     | NULL    |                |
| lname | varchar(30) | YES  |     | NULL    |                |
| city  | varchar(30) | YES  |     | NULL    |                |
| age   | int         | YES  |     | NULL    |                |
| dob   | date        | YES  |     | NULL    |                |
| marks | int         | YES  |     | 0       |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

mysql> INSERT INTO student (fname, lname, city, age, dob, marks) VALUES

('Amit', 'Sharma', 'Delhi', 18, '2006-05-12', 85),
('Riya', 'Verma', 'Mumbai', 17, '2007-08-22', 90),
('Rahul', 'Gupta', 'Kolkata', 19, '2005-11-15', 78),
('Priya', 'Singh', 'Chennai', 18, '2006-02-10', 88),
('Vikram', 'Yadav', 'Bangalore', 20, '2004-07-05', 76),
('Sneha', 'Patel', 'Hyderabad', 17, '2007-03-18', 92),
('Arjun', 'Reddy', 'Pune', 19, '2005-09-25', 81),
('Neha', 'Malhotra', 'Ahmedabad', 18, '2006-12-30', 89),
('Karan', 'Jain', 'Jaipur', 20, '2004-06-14', 74),
('Meera', 'Chopra', 'Lucknow', 19, '2005-10-08', 95);

```
mysql> select *from student;
+-----+--------+----------+-----------+------+------------+-------+
| sid | fname  | lname    | city      | age  | dob        | marks |
+-----+--------+----------+-----------+------+------------+-------+
|   1 | Amit   | Sharma   | Delhi     |   18 | 2006-05-12 |    85 |
|   2 | Riya   | Verma    | Mumbai    |   17 | 2007-08-22 |    90 |
|   3 | Rahul  | Gupta    | Kolkata   |   19 | 2005-11-15 |    78 |
|   4 | Priya  | Singh    | Chennai   |   18 | 2006-02-10 |    88 |
|   5 | Vikram | Yadav    | Bangalore |   20 | 2004-07-05 |    76 |
|   6 | Sneha  | Patel    | Hyderabad |   17 | 2007-03-18 |    92 |
|   7 | Arjun  | Reddy    | Pune      |   19 | 2005-09-25 |    81 |
|   8 | Neha   | Malhotra | Ahmedabad |   18 | 2006-12-30 |    89 |
|   9 | Karan  | Jain     | Jaipur    |   20 | 2004-06-14 |    74 |
|  10 | Meera  | Chopra   | Lucknow   |   19 | 2005-10-08 |    95 |
+-----+--------+----------+-----------+------+------------+-------+
10 rows in set (0.00 sec)

mysql>
```

Query OK, 10 rows affected (0.02 sec)

mysql> select concat("fname", "lname");

mysql> SELECT CONCAT(fname, ' ', lname) AS full_name FROM student;

```
mysql> SELECT CONCAT(fname, ' ', lname) AS full_name FROM student;
+---------------+
| full_name     |
+---------------+
| Amit Sharma   |
| Riya Verma    |
| Rahul Gupta   |
| Priya Singh   |
| Vikram Yadav  |
| Sneha Patel   |
| Arjun Reddy   |
| Neha Malhotra |
| Karan Jain    |
| Meera Chopra  |
+---------------+
10 rows in set (0.00 sec)
```

mysql> select concat_ws("@", "hi", "bye", "tata");

mysql> SELECT CONCAT_WS("--", sid, fname, lname, city, age) FROM student;

```
mysql> SELECT CONCAT_WS("--", sid, fname, lname, city, age) FROM student;
+-----------------------------------------------+
| CONCAT_WS("--", sid, fname, lname, city, age) |
+-----------------------------------------------+
| 1--Amit--Sharma--Delhi--18                    |
| 2--Riya--Verma--Mumbai--17                    |
| 3--Rahul--Gupta--Kolkata--19                  |
| 4--Priya--Singh--Chennai--18                  |
| 5--Vikram--Yadav--Bangalore--20               |
| 6--Sneha--Patel--Hyderabad--17                |
| 7--Arjun--Reddy--Pune--19                     |
| 8--Neha--Malhotra--Ahmedabad--18              |
| 9--Karan--Jain--Jaipur--20                    |
| 10--Meera--Chopra--Lucknow--19                |
+-----------------------------------------------+
10 rows in set (0.00 sec)
```

mysql> SELECT CONCAT_WS(", ", fname, lname, city) AS full_details FROM student;

mysql> SELECT CONCAT_WS(" | ", sid, fname, lname, age, dob, marks) AS student_info FROM student;

mysql> SELECT CONCAT_WS(" - ", "Student", sid, fname, lname, "from", city) AS student_summary FROM student;

```
mysql> SELECT SUBSTRING(dob, 1, 4) FROM student;
+----------------------+
| SUBSTRING(dob, 1, 4) |
+----------------------+
| 2006                 |
| 2007                 |
| 2005                 |
| 2006                 |
| 2004                 |
| 2007                 |
| 2005                 |
| 2006                 |
| 2004                 |
| 2005                 |
+----------------------+
10 rows in set (0.00 sec)

mysql>
```

mysql> SELECT REPLACE(city, 'Delhi', 'New Delhi') FROM student;

```
mysql> SELECT REPLACE(city, 'Delhi', 'New Delhi') FROM student;
+-------------------------------------+
| REPLACE(city, 'Delhi', 'New Delhi') |
+-------------------------------------+
| New Delhi                           |
| Mumbai                              |
| Kolkata                             |
| Chennai                             |
| Bangalore                           |
| Hyderabad                           |
| Pune                                |
| Ahmedabad                           |
| Jaipur                              |
| Lucknow                             |
+-------------------------------------+
10 rows in set (0.00 sec)
```

mysql> SELECT REPLACE(fname, ' ', '_') FROM student;

mysql> SELECT REPLACE(phone, SUBSTRING(phone, 1, 6), 'XXXXXX') FROM student;

1. mysql> select ucase("abdebfajfgag");

Example:

mysql> SELECT UPPER(fname) FROM student;

mysql> SELECT LOWER(lname) FROM student;

mysql> SELECT UPPER(CONCAT(fname, ' ', lname)) FROM student;

mysql> SELECT LOWER(city) FROM student;

mysql> SELECT LOWER(email) FROM student;

```
mysql> SELECT UPPER(fname) FROM student;
+---------------+
| UPPER(fname)  |
+---------------+
| AMIT          |
| RIYA          |
| RAHUL         |
| PRIYA         |
| VIKRAM        |
| SNEHA         |
| ARJUN         |
| NEHA          |
| KARAN         |
| MEERA         |
+---------------+
10 rows in set (0.00 sec)
```

mysql> SELECT fname, CHAR_LENGTH(fname) AS length_of_name FROM student;

```
mysql> SELECT fname, CHAR_LENGTH(fname) AS length_of_name FROM student;
+--------+-----------------+
| fname  | length_of_name  |
+--------+-----------------+
| Amit   |               4 |
| Riya   |               4 |
| Rahul  |               5 |
| Priya  |               5 |
| Vikram |               6 |
| Sneha  |               5 |
| Arjun  |               5 |
| Neha   |               4 |
| Karan  |               5 |
| Meera  |               5 |
+--------+-----------------+
10 rows in set (0.00 sec)
```

mysql>SELECT fname, lname, CHAR_LENGTH(CONCAT(fname, ' ', lname)) AS length_of_full_name FROM student;

mysql>SELECT city, CHAR_LENGTH(city) AS length_of_city FROM student;

mysql> SELECT sid, fname, CHAR_LENGTH(CONCAT(sid, fname)) AS length_of_sid_fname FROM student;

DATE FUNCTIONS

Function	Description	Example
NOW()	Returns the current timestamp.	SELECT NOW();
CURDATE()	Returns the current date.	SELECT CURDATE();
CURTIME()	Returns the current time.	SELECT CURTIME();
DATE_FORMAT()	Formats a date.	SELECT DATE_FORMAT(NOW(), '%Y-%m-%d');

mysql> select curdate(); mysql> select curtime(); mysql> select now();

mysql> create table emp_det(joj date, jojt time, jojdt datetime);

mysql> desc emp_det; mysql> insert into emp_det values(curdate(), curtime(), now());

mysql> insert into emp_det values(curdate(), curtime(), now());

mysql> desc emp_det; mysql> select *from emp_det;

mysql> select dayname("2000-08-20"); mysql> select dayofmonth("2000-08-20");

mysql> select month("2000-08-20"); mysql> select date_format(now(), "%d/%m/%y");

mysql> select date_format(now(), "%d-%m-%y");

mysql> select date_format(now(), "%d %a at %k");

mysql> select date_format(now(), "%d %a at %k");

DATE_FORMAT() Formats in MySQL

Format Description	Query	Example Output
Full Date (YYYY-MM-DD)	SELECT DATE_FORMAT(NOW(), '%Y-%m-%d');	2/10/2025
Full Date with Time	SELECT DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s');	2/10/2025 14:30
Day-Month-Year	SELECT DATE_FORMAT(NOW(), '%d-%m-%Y');	10/2/2025
Month-Day-Year	SELECT DATE_FORMAT(NOW(), '%m-%d-%Y');	2/10/2025
Day/Month/Year	SELECT DATE_FORMAT(NOW(), '%d/%m/%Y');	10/2/2025
Month Name, Day Year	SELECT DATE_FORMAT(NOW(), '%M %d, %Y');	10-Feb-25
Short Month Name, Day Year	SELECT DATE_FORMAT(NOW(), '%b %d, %Y');	10-Feb-25
Weekday, Month Day, Year	SELECT DATE_FORMAT(NOW(), '%W, %M %d, %Y');	Monday, February 10, 2025
Time in 12-hour Format	SELECT DATE_FORMAT(NOW(), '%r');	2:30:15 PM
Time in 24-hour Format	SELECT DATE_FORMAT(NOW(), '%H:%i:%s');	14:30:15
Hour:Minute AM/PM	SELECT DATE_FORMAT(NOW(), '%h:%i %p');	2:30 PM
Day Name Only	SELECT DATE_FORMAT(NOW(), '%W');	Monday
Month Name Only	SELECT DATE_FORMAT(NOW(), '%M');	February
Year Only (4 Digits)	SELECT DATE_FORMAT(NOW(), '%Y');	2025
Year Only (2 Digits)	SELECT DATE_FORMAT(NOW(), '%y');	25
Month Number (01-12)	SELECT DATE_FORMAT(NOW(), '%m');	2
Day of the Month (01-31)	SELECT DATE_FORMAT(NOW(), '%d');	10
Day of the Year (001-366)	SELECT DATE_FORMAT(NOW(), '%j');	41
Week Number (00-53)	SELECT DATE_FORMAT(NOW(), '%U');	6
ISO Week Number (00-53)	SELECT DATE_FORMAT(NOW(), '%V');	7

JOIN

JOIN is used to combine rows from two or more tables based on a related column between them.

Types of Joins

1. **INNER JOIN** – Returns matching rows from both tables.
2. **LEFT JOIN (or LEFT OUTER JOIN)** – Returns all rows from the left table and matching rows from the right table.
3. **RIGHT JOIN (or RIGHT OUTER JOIN)** – Returns all rows from the right table and matching rows from the left table.
4. **FULL JOIN (or FULL OUTER JOIN**) – Not directly supported in MySQL but can be achieved using UNION.
5. **CROSS JOIN** – Returns the Cartesian product of both tables.

❖ Example of INNER JOIN

Step 1: Create Two Tables

```
CREATE TABLE students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);

CREATE TABLE courses (
   course_id INT PRIMARY KEY,
   student_id INT,
   course_name VARCHAR(50),
   FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

```
mysql> desc students;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| student_id | int         | NO   | PRI | NULL    |       |
| name       | varchar(50) | YES  |     | NULL    |       |
| age        | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> desc course;
ERROR 1146 (42S02): Table 'rid.course' doesn't exist
mysql> desc courses;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | int         | NO   | PRI | NULL    |       |
| student_id  | int         | YES  | MUL | NULL    |       |
| course_name | varchar(50) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

Step 2: Insert Data

```
INSERT INTO students VALUES (1, 'Amit', 18), (2, 'Rahul', 19), (3, 'Priya', 18);
INSERT INTO courses VALUES (101, 1, 'Mathematics'), (102, 2, 'Science');
```

Step 3: Use INNER JOIN

```
SELECT students.student_id, students.name,
courses.course_name
FROM students
INNER JOIN courses ON students.student_id =
courses.student_id;
```

```
mysql> SELECT students.student_id, students.name, courses.course_name
    -> FROM students
    -> INNER JOIN courses ON students.student_id = courses.student_id;
+------------+-------+-------------+
| student_id | name  | course_name |
+------------+-------+-------------+
|          1 | Amit  | Mathematics |
|          2 | Rahul | Science     |
+------------+-------+-------------+
2 rows in set (0.00 sec)
```

```
mysql> select *from students;
+------------+-------+------+
| student_id | name  | age  |
+------------+-------+------+
|          1 | Amit  |   18 |
|          2 | Rahul |   19 |
|          3 | Priya |   18 |
+------------+-------+------+
3 rows in set (0.00 sec)

mysql> select *from courses;
+-----------+------------+-------------+
| course_id | student_id | course_name |
+-----------+------------+-------------+
|       101 |          1 | Mathematics |
|       102 |          2 | Science     |
+-----------+------------+-------------+
2 rows in set (0.00 sec)
```

1. INNER JOIN

- **INNER JOIN** returns only the rows where there is a match in both tables based on a common column.
- If there is no match, the row is **not included** in the result.
- **Syntax:** SELECT table1.column_name, table2.column_name FROM table1 INNER JOIN table2 ON table1.common_column = table2.common_column;

Example-2:

Step-1 Create Two Tables (employees and departments)

```
CREATE TABLE employees (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR(50),
  dept_id INT
);
CREATE TABLE departments (
  dept_id INT PRIMARY KEY,
  dept_name VARCHAR(50)
);
```

```
mysql> desc employees;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int         | NO   | PRI | NULL    |       |
| emp_name | varchar(50) | YES  |     | NULL    |       |
| dept_id  | int         | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc departments;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| dept_id   | int         | NO   | PRI | NULL    |       |
| dept_name | varchar(50) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00
```

Step-2: Insert Sample Data

```
INSERT INTO employees VALUES
(1, 'Amit', 101),
(2, 'Rahul', 102),
(3, 'Priya', 103),
(4, 'Suresh', 101);
INSERT INTO departments VALUES
(101, 'HR'),
(102, 'Finance'),
(104, 'IT');
```

```
mysql> select *from employees;
+--------+----------+---------+
| emp_id | emp_name | dept_id |
+--------+----------+---------+
|      1 | Amit     |     101 |
|      2 | Rahul    |     102 |
|      3 | Priya    |     103 |
|      4 | Suresh   |     101 |
+--------+----------+---------+
4 rows in set (0.00 sec)

mysql> select *from departments;
+---------+-----------+
| dept_id | dept_name |
+---------+-----------+
|     101 | HR        |
|     102 | Finance   |
|     104 | IT        |
+---------+-----------+
3 rows in set (0.00 sec)
```

Step-3: join the table

```
SELECT employees.emp_id, employees.emp_name, departments.dept_name
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

2. LEFT JOIN & RIGHT JOIN

❖ LEFT JOIN:

- Returns all rows from the left table (employees) and matching rows from the right table (departments). If there is no match in the right table, NULL is returned.
- **Syntax:**

```
SELECT employees.emp_id, employees.emp_name, departments.dept_name
FROM employees
LEFT JOIN departments ON employees.dept_id = departments.dept_id;
```

Example:

```
mysql> SELECT employees.emp_id, employees.emp_name, departments.dept_name
    -> FROM employees
    -> LEFT JOIN departments ON employees.dept_id = departments.dept_id;
+--------+----------+-----------+
| emp_id | emp_name | dept_name |
+--------+----------+-----------+
|      1 | Amit     | HR        |
|      2 | Rahul    | Finance   |
|      3 | Priya    | NULL      |
|      4 | Suresh   | HR        |
+--------+----------+-----------+
4 rows in set (0.00 sec)
```

Explanation:

- All employees are displayed.
- Priya (dept_id = 103) has **no matching department**, so NULL appears in the dept_name column.
- The **IT department (dept_id = 104)** is **not shown** because it does not match any employee.

❖ RIGHT JOIN (or RIGHT OUTER JOIN)

Returns all rows from the right table (**departments**) and matching rows from the left table (**employees**). If there is no match in the left table, NULL is returned.

Syntax:
```
SELECT employees.emp_id, employees.emp_name, departments.dept_name
FROM employees
RIGHT JOIN departments ON employees.dept_id = departments.dept_id;
```

```
mysql> SELECT employees.emp_id, employees.emp_name, departments.dept_name
    -> FROM employees
    -> RIGHT JOIN departments ON employees.dept_id = departments.dept_id;
+--------+----------+-----------+
| emp_id | emp_name | dept_name |
+--------+----------+-----------+
|      4 | Suresh   | HR        |
|      1 | Amit     | HR        |
|      2 | Rahul    | Finance   |
|   NULL | NULL     | IT        |
+--------+----------+-----------+
4 rows in set (0.00 sec)
```

3. FULL OUTER JOIN & CROSS JOIN

❖ FULL OUTER JOIN (Using UNION)

- **MySQL does not support FULL OUTER JOIN directly**, but we can achieve the same result using **LEFT JOIN + RIGHT JOIN + UNION**.
- It **returns all records** from both tables, showing NULL when there is no match.
- **Syntax:**

```
SELECT employees.emp_id, employees.emp_name, departments.dept_name
FROM employees
LEFT JOIN departments ON employees.dept_id = departments.dept_id
UNION
SELECT employees.emp_id, employees.emp_name, departments.dept_name
FROM employees
RIGHT JOIN departments ON employees.dept_id = departments.dept_id;
```

Example:

```
mysql> SELECT employees.emp_id, employees.emp_name, departments.dept_name
    -> FROM employees
    -> LEFT JOIN departments ON employees.dept_id = departments.dept_id
    ->
    -> UNION
    ->
    -> SELECT employees.emp_id, employees.emp_name, departments.dept_name
    -> FROM employees
    -> RIGHT JOIN departments ON employees.dept_id = departments.dept_id;
+--------+----------+-----------+
| emp_id | emp_name | dept_name |
+--------+----------+-----------+
|      1 | Amit     | HR        |
|      2 | Rahul    | Finance   |
|      3 | Priya    | NULL      |
|      4 | Suresh   | HR        |
|   NULL | NULL     | IT        |
+--------+----------+-----------+
```

Explanation:

- Combines **LEFT JOIN** and **RIGHT JOIN** results.
- Shows all employees, even if they don't belong to any department (e.g., Priya).
- Shows all departments, even if no employee is assigned (e.g., IT department).

UNION

- UNION is used to combine the result sets of two or more SELECT queries.
- It **removes duplicate rows** by default.

❖ **Syntax:** SELECT column_name FROM table1 **UNION** SELECT column_name FROM table2;

❖ **Example:**

- **Create Two Tables (students & teachers)**

  CREATE TABLE **studentss** (id INT PRIMARY KEY, name VARCHAR(50));
  CREATE TABLE **teachers** (id INT PRIMARY KEY, name VARCHAR(50));

- **Insert Sample Data**

  INSERT INTO studentss VALUES (1, 'Amit'), (2, 'Rahul'), (3, 'Priya');
  INSERT INTO teachers VALUES (101, 'Mr. Sharma'), (102, 'Ms. Verma');

❖ **Use UNION to Combine Results**

SELECT name FROM students UNION SELECT name FROM teachers;

```
mysql> select *from students;
+----+-------+
| id | name  |
+----+-------+
|  1 | Amit  |
|  2 | Rahul |
|  3 | Priya |
+----+-------+
3 rows in set (0.00 sec)

mysql> select *from teachers;
+-----+------------+
| id  | name       |
+-----+------------+
| 101 | Mr. Sharma |
| 102 | Ms. Verma  |
+-----+------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT name FROM students
    -> UNION
    -> SELECT name FROM teachers;
+------------+
| name       |
+------------+
| Amit       |
| Rahul      |
| Priya      |
| Mr. Sharma |
| Ms. Verma  |
+------------+
5 rows in set (0.00 sec)
```

INTERSECTION

- MySQL **does not have a direct INTERSECT operator** like other databases (PostgreSQL, SQL Server).
- However, we can achieve **INTERSECTION** using **INNER JOIN** or **UNION ALL with GROUP BY**.

1. Using INNER JOIN (Recommended) → We find common records in two tables using an **INNER JOIN**.

Example:

```
SELECT students.name
FROM students
INNER JOIN teachers ON students.name = teachers.name;
```

- This will return names that **exist in both** students and teachers tables.

2. Using UNION ALL with GROUP BY

- If we need an INTERSECT operation without using JOIN, we can use UNION ALL with GROUP BY and HAVING COUNT(*) > 1.

Example: SELECT name FROM (SELECT name FROM students UNION ALL
SELECT name FROM teachers) AS combinedGROUP BY name HAVING COUNT(name) > 1;

- This method finds names that appear in **both tables**.

❖ **Key Points:**

- **INNER JOIN is the best method** for intersection.
- UNION ALL with GROUP BY is useful if the tables don't have a common key.

Example: - for intersection Create student's table

```
CREATE TABLE students (
   name VARCHAR(50),
   class INT
);

-- Insert sample data
INSERT INTO students (name, class) VALUES
('Tanmay', 10),
('Rajesh', 12),
('Ankit', 11),
('Priya', 9);

-- Create teachers table
CREATE TABLE teachers (
   name VARCHAR(50),
   subject VARCHAR(50)
);

-- Insert sample data
INSERT INTO teachers (name, subject) VALUES
('Rajesh', 'Physics'),
('Sneha', 'English'),
('Tanmay', 'Computer'),
('Amit', 'Math');

-- INTERSECTION using INNER JOIN
SELECT students.name
FROM students
INNER JOIN teachers
ON students.name = teachers.name;
```

TRANSACTIONS

Transaction in MySQL is a set of SQL queries that must be executed **together** as a unit. If all queries succeed,changes are saved (**COMMIT**), but if any query fails, all changes are **rolled back** (**ROLLBACK**).

❖ Why Use Transactions?

- Ensures **data consistency** (e.g., money transfer between accounts).
- Prevents **partial updates** (e.g., updating one table but not another).
- Used in **banking, inventory management, and multi-step operations**.

❖ Transaction Properties (ACID)

- Transactions follow **ACID** rules:
  1. **Atomicity** – All queries execute completely or none at all.
  2. **Consistency** – The database remains valid before and after the transaction.
  3. **Isolation** – Transactions run independently without affecting each other.
  4. **Durability** – Once committed, data is permanently saved.

Example: Money Transfer Between Two Accounts

Step 1: Create the Table

```
CREATE TABLE Accounts (
   AccountID INT PRIMARY KEY,
   Name VARCHAR(50),
   Balance DECIMAL(10,2)   );
```

Step 2: Insert Sample Data

```
INSERT INTO Accounts (AccountID, Name, Balance) VALUES
(1, 'Aman', 5000.00),
(2, 'Riya', 3000.00);
```

```
mysql> select *from  accounts;
+-----------+------+---------+
| AccountID | Name | Balance |
+-----------+------+---------+
|         1 | Aman | 5000.00 |
|         2 | Riya | 3000.00 |
+-----------+------+---------+
2 rows in set (0.00 sec)
```

Step 3: Transfer ₹1000 from Aman to Riya using a Transaction

```
START TRANSACTION;
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 1; -- Deduct from Aman
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountID = 2; -- Add to Riya
```

```
mysql> select *from accounts;
+-----------+------+---------+
| AccountID | Name | Balance |
+-----------+------+---------+
|         1 | Aman | 4000.00 |
|         2 | Riya | 4000.00 |
+-----------+------+---------+
2 rows in set (0.00 sec)
```

Note: START TRANSACTION; it is **not mandatory** to use

❖ When is START TRANSACTION; Required?

1. **Without Transaction** – Each UPDATE runs **independently,** and changes are permanent immediately.
2. **With Transaction (START TRANSACTION;)** – Changes are only saved if COMMIT; is executed. If an error occurs, we can undo changes with ROLLBACK;.

Example Without START TRANSACTION; (Each Query Executes Immediately)

```
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 1; -- Deduct ₹1000 from Aman
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountID = 2; -- Add ₹1000 to Riya
```

Note: if the second UPDATE fails, Aman still loses ₹1000, causing data inconsistency.

Example With START TRANSACTION; (Ensures Data Consistency)

```
START TRANSACTION;
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountID = 2;
COMMIT; -- Saves changes
```

Note: if any query fails, the transaction is not saved until COMMIT;

❖ What Happens If an Error Occurs?

- **Without START TRANSACTION;** – If the first update succeeds but the second fails, **Aman loses money, but Riya doesn't receive it** (Data inconsistency).
  With START TRANSACTION; – If the second query fails, we can **ROLLBACK** to undo all changes (Data remains consistent).

❖ When to Use `START TRANSACTION;`

- Use transactions (`START TRANSACTION;`) when multiple queries must be executed together
- For operations like money transfer, stock updates, and multi-step changes
- If one query failing should undo all previous updates, transactions are required.

❖ Conclusion

- Not mandatory for simple UPDATE queries.
- Mandatory if you want atomicity (all or nothing execution).
- Highly recommended for financial transactions, inventory updates, and dependent operations.

Note: IF...THEN...END IF; statement **cannot be used directly in MySQL at the command line or in a normal transaction block**. It can only be used inside a **stored procedure, function, or a BEGIN...END block**.

NORMALIZATION

it is a process in database design that organizes data to reduce redundancy and improve data integrity. It involves dividing large tables into smaller tables and defining relationships between them.

❖ Why Normalization?

1. **Eliminates Data Redundancy –** Reduces duplicate data.
2. **Improves Data Integrity –** Ensures that data remains accurate and consistent.
3. **Enhances Query Performance –** Efficiently organizes data for faster retrieval.
4. **Reduces Update Anomalies –** Prevents issues when inserting, updating, or deleting data.

❖ Types of Normalization in MySQL

1. **First Normal Form (1NF) –** Eliminates duplicate columns and ensures atomicity.
2. **Second Normal Form (2NF) –** Removes partial dependencies.
3. **Third Normal Form (3NF) –** Eliminates transitive dependencies.
4. **Boyce-Codd Normal Form (BCNF)**stricter version of 3NF, ensuring every determinant is a candidate key.
5. **Fourth Normal Form (4NF) –** Removes multi-valued dependencies.
6. **Fifth Normal Form (5NF) –** Eliminates join dependencies.

❖ Normalization Forms

- Normalization is done in multiple stages, called Normal Forms (NF). Each higher normal form removes more redundancy and improves data consistency.

1NF (FIRST NORMAL FORM)

1NF (First Normal Form) – Remove Duplicate Columns

A table is in First Normal Form (1NF) if:

- Each column contains atomic (indivisible) values.
- Each column contains values of a single type.
- The table has a primary key to identify each row uniquely.

Example (Before 1NF - Unnormalized Table)

StudentID	Name	Subjects
1	Amit	Math, Science
2	Raj	English, Math
3	Priya	Science

Here, the **Subjects** column contains multiple values, violating 1NF.

After 1NF (Atomic Data)

StudentID	Name	Subject
1	Amit	Math
1	Amit	Science
2	Raj	English
2	Raj	Math
3	Priya	Science

Step 1: Create an Unnormalized Table (Before 1NF)

- Since the table is unnormalized, the Subjects column contains multiple values in a single cell, which is incorrect.

SQL Query to Create the Unnormalized Table

```
CREATE DATABASE NDB;
USE NDB;
CREATE TABLE Students_Un (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(50),
   Subjects VARCHAR(100) -- Multiple subjects stored in one column (violating 1NF)
);
INSERT INTO Students_Un (StudentID, Name, Subjects) VALUES
(1, 'Amit', 'Math, Science'),
(2, 'Raj', 'English, Math'),
(3, 'Priya', 'Science');
-- Display the table
SELECT * FROM Students_Unnormalized;
```

Output (Before 1NF - Unnormalized Table)

StudentID	Name	Subjects
1	Amit	Math, Science
2	Raj	English, Math
3	Priya	Science

Step 2: Create a New Table in 1NF

- To achieve 1NF, we need to split the Subjects column so that each subject gets a separate row.

SQL Query to Create the 1NF Table

```
CREATE TABLE Students_1NF (
    StudentID INT,
    Name VARCHAR(50),
    Subject VARCHAR(50),
    PRIMARY KEY (StudentID, Subject));
 -- Composite Primary Key);
INSERT INTO Students_1NF
(StudentID, Name, Subject) VALUES
(1, 'Amit', 'Math'),
(1, 'Amit', 'Science'),
(2, 'Raj', 'English'),
(2, 'Raj', 'Math'),
(3, 'Priya', 'Science');
-- Display the table
SELECT * FROM Students_1NF;
```

Step 3: Explanation of the Changes

What We Fixed?

1. **Atomicity** – Each column now contains only a single value.
2. **No Repeating Groups** – Each subject has a separate row.
3. **Elimination of Data Redundancy** – Data is structured properly for easy querying.

Why is This Useful?

- Now, we can easily filter students by subject.
- It follows relational database best practices.
- Avoids data anomalies in insertion, update, and deletion.

Output (After 1NF - Atomic Data)

StudentID	Name	Subject
1	Amit	Math
1	Amit	Science
2	Raj	English
2	Raj	Math
3	Priya	Science

2NF (Second Normal Form)

2NF (Second Normal Form) – Remove Partial Dependencies

A table is in **Second Normal Form (2NF)** if:

✅ It is in **1NF**.

✅ **No partial dependency** exists (i.e., non-key attributes should depend on the whole primary key, not just part of it).

Example (Before 2NF - Partial Dependency)

Consider a composite key:

StudentID	CourseID	StudentName	CourseName
1	C101	Amit	Math
2	C102	Raj	Science

Here, **StudentName** depends only on **StudentID**, and **CourseName** depends only on **CourseID**. This violates 2NF.

After 2NF (Breaking into Two Tables)

1. **Students Table**

StudentID	StudentName
1	Amit
2	Raj

2. **Courses Table**

CourseID	CourseName
C101	Math
C102	Science

3. **Enrollment Table** (to establish relationships)

StudentID	CourseID
1	C101
2	C102

Now, **StudentName** depends only on **StudentID**, and **CourseName** depends only on **CourseID**.

Step 1: Create the Table in 1NF (Before 2NF)

SQL Query to Create the 1NF Table

```
CREATE DATABASE NDB;
USE NDB;
CREATE TABLE Students_1NF (
   StudentID INT,
   CourseID VARCHAR(10),
   StudentName VARCHAR(50),
   CourseName VARCHAR(50),
   PRIMARY KEY (StudentID, CourseID)  -- Composite Primary Key
);
INSERT INTO Students_1NF (StudentID, CourseID, StudentName, CourseName) VALUES
(1, 'C101', 'Amit', 'Math'),
(2, 'C102', 'Raj', 'Science');  -- Display the table
SELECT * FROM Students_1NF;
```

Output (Before 2NF - Partial Dependency)

StudentID	CourseID	StudentName	CourseName
1	C101	Amit	Math
2	C102	Raj	Science

Problems in this Table:

- Partial Dependency:
  - `StudentName` depends **only on** `StudentID` (not on `CourseID`).
  - `CourseName` depends **only on** `CourseID` (not on `StudentID`).
- This violates **2NF**, as non-key attributes should depend on the whole primary key, not just a part of it.

Step 2: Convert the Table into 2NF

To eliminate partial dependency, we split the table into three separate tables:

1. **Students Table –** Contains StudentID and StudentName.
2. **Courses Table –** Contains CourseID and CourseName.
3. **Enrollment Table –** Establishes a many-to-many relationship between Students and Courses.

SQL Queries to Create the 2NF Tables

1. Create Students Table

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   StudentName VARCHAR(50));
INSERT INTO Students (StudentID, StudentName) VALUES
(1, 'Amit'),
(2, 'Raj');  -- Display Students Table
SELECT * FROM Students;
```

2. Create Courses Table

```
CREATE TABLE Courses (
   CourseID VARCHAR(10) PRIMARY KEY,
   CourseName VARCHAR(50)
);
```

```
INSERT INTO Courses (CourseID, CourseName) VALUES
('C101', 'Math'),
('C102', 'Science'); -- Display Courses Table
SELECT * FROM Courses;
```

3. Create Enrollment Table (Many-to-Many Relationship)

```
CREATE TABLE Enrollment (
   StudentID INT,
   CourseID VARCHAR(10),
   PRIMARY KEY (StudentID, CourseID),
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
INSERT INTO Enrollment (StudentID, CourseID) VALUES
(1, 'C101'),
(2, 'C102');
SELECT * FROM Enrollment;
```

Step 3: Verify the Tables After 2NF

Now, if we query all three tables, we will get:

Students Table

StudentID	StudentName
1	Amit
2	Raj

Courses Table

CourseID	CourseName
C101	Math
C102	Science

Enrollment Table

StudentID	CourseID
1	C101
2	C102

3NF (Third Normal Form)

3. Third Normal Form (3NF) – Eliminates Transitive Dependency

A table is in **3NF** if:

It is in **2NF**.

No **transitive dependency** exists (i.e., non-key attributes should depend only on the primary key).

Example (Before 3NF - Transitive Dependency)

StudentID	StudentName	CourseID	CourseName	Instructor	InstructorOffice
1	Amit	C101	Math	Dr. A	Room 101
2	Raj	C102	Science	Dr. B	Room 102

Here, **InstructorOffice** depends on **Instructor**, which depends on **CourseID** (not directly on StudentID).

Step 4: Explanation of the Changes

✅ **What We Fixed?**

- We **removed partial dependencies** by creating separate tables for `Students` and `Courses`.
- The **Enrollment Table** now only stores **StudentID** and **CourseID**, keeping data relationships intact.
- **Data Redundancy is Reduced:** `StudentName` and `CourseName` are stored only once in their respective tables.

✅ **Why is This Useful?**

- Now, we can **easily add new courses** without duplicating student data.
- We can **add new students** without modifying the course information.
- This follows **relational database best practices** and prevents update anomalies.

After 3NF (Creating a Separate Table)

1. **Instructors Table**

Instructor	InstructorOffice
Dr. A	Room 101
Dr. B	Room 102

2. **Courses Table**

CourseID	CourseName	Instructor
C101	Math	Dr. A
C102	Science	Dr. B

Step 1: Create the Table in 2NF (Before 3NF)

The table contains a transitive dependency:

- InstructorOffice depends on Instructor, which in turn depends on CourseID (not directly on StudentID).
- This violates 3NF, as non-key attributes should depend only on the primary key.

SQL Query to Create the 2NF Table (Before 3NF)

```
CREATE DATABASE NDB;
USE NDB;
CREATE TABLE Students_2NF (
   StudentID INT,
   StudentName VARCHAR(50),
   CourseID VARCHAR(10),
   CourseName VARCHAR(50),
   Instructor VARCHAR(50),
   InstructorOffice VARCHAR(50),
   PRIMARY KEY (StudentID, CourseID)
);
INSERT INTO Students_2NF (StudentID, StudentName, CourseID, CourseName, Instructor, InstructorOffice) VALUES
(1, 'Amit', 'C101', 'Math', 'Dr. A', 'Room 101'),
(2, 'Raj', 'C102', 'Science', 'Dr. B', 'Room 102');-- Display the table
SELECT * FROM Students_2NF;
```

Output (Before 3NF - Transitive Dependency)

StudentID	StudentName	CourseID	CourseName	Instructor	InstructorOffice
1	Amit	C101	Math	Dr. A	Room 101
2	Raj	C102	Science	Dr. B	Room 102

Step 2: Convert the Table into 3NF

To eliminate the **transitive dependency**, we split the table into three separate tables:

1. **Instructors Table** – Contains Instructor and InstructorOffice.
2. **Courses Table** – Contains CourseID, CourseName, and Instructor.
3. **Students Table** – Contains StudentID and StudentName.
4. **Enrollment Table** – Establishes the relationship between Students and Courses.

SQL Queries to Create the 3NF Tables

1. Create Students Table

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   StudentName VARCHAR(50)
);
INSERT INTO Students (StudentID, StudentName) VALUES
(1, 'Amit'),
(2, 'Raj');  -- Display Students Table
SELECT * FROM Students;
```

2. Create Instructors Table

```
CREATE TABLE Instructors (
   Instructor VARCHAR(50) PRIMARY KEY,
   InstructorOffice VARCHAR(50)
);
INSERT INTO Instructors (Instructor, InstructorOffice) VALUES
('Dr. A', 'Room 101'),
('Dr. B', 'Room 102');  -- Display Instructors Table
SELECT * FROM Instructors;
```

3. Create Courses Table

```
CREATE TABLE Courses (
   CourseID VARCHAR(10) PRIMARY KEY,
   CourseName VARCHAR(50),
   Instructor VARCHAR(50),
   FOREIGN KEY (Instructor) REFERENCES Instructors(Instructor)
);
INSERT INTO Courses (CourseID, CourseName, Instructor) VALUES
('C101', 'Math', 'Dr. A'),
('C102', 'Science', 'Dr. B');  -- Display Courses Table
SELECT * FROM Courses;
```

4. Create Enrollment Table

```
CREATE TABLE Enrollment (
   StudentID INT,
   CourseID VARCHAR(10),
   PRIMARY KEY (StudentID, CourseID),
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
INSERT INTO Enrollment (StudentID, CourseID) VALUES
(1, 'C101'),
(2, 'C102'); -- Display Enrollment Table
SELECT * FROM Enrollment;
```

Students Table

StudentID	StudentName
1	Amit
2	Raj

Instructors Table

Instructor	InstructorOffice
Dr. A	Room 101
Dr. B	Room 102

Courses Table

CourseID	CourseName	Instructor
C101	Math	Dr. A
C102	Science	Dr. B

Enrollment Table

StudentID	CourseID
1	C101
2	C102

Step 4: Explanation of the Changes

What We Fixed?

- Removed Transitive Dependency:
  - `InstructorOffice` no longer depends on `Instructor` via `CourseID`.
  - Now, `InstructorOffice` is stored separately in the `Instructors` table.
- Data Redundancy is Reduced:
  - Instructor details are stored only once, making updates easier.

Why is This Useful?

- We can easily update instructor information without modifying the courses.
- We can add new courses or students without redundant data.
- This follows relational database best practices and ensures data consistency.

4.BCNF (Boyce-Codd Normal Form)

4. Boyce-Codd Normal Form (BCNF) – Stronger than 3NF

A table is in **BCNF** if:

✅ It is in **3NF**.

✅ Every **determinant** is a **candidate key**.

eterminant is an attribute that uniquely determines another attribute.

Example (Before BCNF - Violation of Dependency Rule)

StudentID	CourseID	Instructor
1	C101	Dr. A
2	C101	Dr. A
3	C102	Dr. B

Here, **CourseID** determines **Instructor**, but **CourseID** is not a candidate key, violating BCNF.

After BCNF (Splitting the Table)

1. **Courses Table**

CourseID	Instructor
C101	Dr. A
C102	Dr. B

2. **Enrollment Table**

StudentID	CourseID
1	C101
2	C101
3	C102

Example 1: Single Determinant Consider a **Students table**:

Roll_No (PK)	Student_Name	Class
101	Raj	10th
102	Ankit	9th

👉 **Here, Roll_No is a Determinant because:**

- Roll_No is **unique** and **determines both Student_Name and Class**.
- If we know Roll_No, we can find the corresponding student's name

Example 2: Composite Determinant

Consider a Marks table:

Roll_No	Subject	Marks
101	Math	85
101	Science	78
102	Math	90

Here, (Roll_No, Subject) together form a Composite Determinant because:

- Marks cannot be determined by Roll_No alone, since a student has multiple subjects.
- But the combination of Roll_No and Subject uniquely determines the Marks for that student in that subject.

Note:

- ✓ A Determinant is a column (or set of columns) that determines the value of another column.
- ✓ A Primary Key is always a Determinant, but not all Determinants are Primary Keys.
- ✓ A Composite Determinant consists of two or more columns that together determine another attribute.

4NF (Fourth Normal Form)

5. Fourth Normal Form (4NF) – Eliminates Multi-Valued Dependencies

A table is in **4NF** if:

It is in **BCNF**.

It has no **multi-valued dependencies** (i.e., when one column contains multiple independent values for the same key).

Example (Before 4NF - Multi-Valued Dependency)

EmployeeID	Skills	Languages
1	Java	English
1	Python	English
1	Java	Hindi

Here, **Skills** and **Languages** are independent but stored in the same table, violating 4NF.

6. Fifth Normal Form (5NF) – Eliminates Join Dependencies

A table is in **5NF** if:

It is in **4NF**.

It has no **join dependencies** (i.e., when data is split across multiple tables and requires complex joins to reconstruct).

This form is useful for complex business applications where multiple relationships exist among attributes.

TRIGGER

Trigger in MySQL is a database object that is automatically executed (or fired) when a specific event occurs in a table. It is mainly used to enforce business rules, maintain audit logs, or update related tables automatically.

❖ Types of Triggers in MySQL:

- Triggers in MySQL are categorized based on the event (INSERT, UPDATE, DELETE) and the execution timing (BEFORE, AFTER):

1. **BEFORE INSERT –** Executes before a new row is inserted into a table.
2. **AFTER INSERT –** Executes after a new row is inserted.
3. **BEFORE UPDATE –** Executes before an existing row is updated.
4. **AFTER UPDATE –** Executes after an existing row is updated.
5. **BEFORE DELETE –** Executes before a row is deleted.
6. **AFTER DELETE –** Executes after a row is deleted.

Example: Creating a Trigger in MySQL Scenario:

- We have an employees table, and we want to maintain a log whenever a new employee is added. We create a trigger that automatically inserts data into the employee_log table after inserting a new employee.

Step 1: Create Tables

```
CREATE TABLE employees (
emp_id INT AUTO_INCREMENT PRIMARY KEY,
 emp_name VARCHAR(50),
salary DECIMAL(10,2));
```

```
CREATE TABLE employee_log (
        log_id INT AUTO_INCREMENT PRIMARY KEY,
        emp_id INT,
        action VARCHAR(50),
        log_time TIMESTAMP DEFAULT
      CURRENT_TIMESTAMP);
```

Step 2: Create a Trigger

```
DELIMITER $$
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employee_log (emp_id, action)
   VALUES (NEW.emp_id, 'Employee Added');
END $$
DELIMITER ;
```

Explanation:

- The AFTER INSERT trigger fires after a new row is added to the employees table.
- The NEW.emp_id refers to the emp_id of the newly inserted employee.
- It inserts a log entry into the employee_log table.

Step 3: Insert Data and See the Trigger in Action

- INSERT INTO employees (emp_name, salary) VALUES ('John Doe', 50000.00);
- Now, check the employee_log table: **SELECT * FROM employee_log;**

❖ Advantages of Triggers in MySQL

1. **Automatic Execution:** No need to manually run queries to maintain logs or enforce rules.
2. **Data Integrity:** Ensures consistency between related tables.
3. **Audit Logs:** Automatically keeps track of changes in a table.
4. **Prevention of Invalid Transactions:** Can be used to validate data before changes occur.

ASCENDING ORDER AND DESCENDING ORDER

1. Ascending Order (ASC)

- The default sorting order in MySQL.
- Sorts values from **lowest to highest** (A → Z, 0 → 9).
- **Example:** SELECT * FROM employees ORDER BY salary ASC;
- **Result:** Employees sorted in **increasing** order of salary.

2. Descending Order (DESC) Sorts values from **highest to lowest** (Z → A, 9 → 0).

- **Example:** SELECT * FROM employees ORDER BY salary DESC;
- **Result:** Employees sorted in **decreasing** order of salary.

Example Table: employees

Query 1: Sort by salary in Ascending Order SELECT * FROM employees ORDER BY salary ASC;

Query 2: Sort by salary in Descending Order SELECT * FROM employees ORDER BY salary DESC;

- **Sorting by Multiple Columns**
- If two employees have the same salary, you can **sort by another column**.
- **Example:** Sort by salary (descending), then emp_name (ascending).
- **SELECT * FROM employees ORDER BY salary DESC, emp_name ASC;**

Conclusion**ASC (Ascending):** Smallest → Largest (Default).**DESC (Descending):** Largest → Smallest.

Multiple Column Sorting: Useful for breaking ties.

How to rename the database

cannot rename a database directly using a MODIFY DATABASE or RENAME DATABASE command. Instead, you need to create a new database, move the data, and then drop the old database.

Follow these steps:

1. Create a New Database **CREATE DATABASE newdb;**

2. Copy All Tables from nredb to newdb → USE nredb; → SHOW TABLES;

For each table, run:

CREATE TABLE newdb.table_name LIKE nredb.table_name;

INSERT INTO newdb.table_name SELECT * FROM nredb.table_name;

3. Drop the Old Database (After Verifying Data in newdb)

DROP DATABASE nredb;

4. Use the New Database → USE newdb;

Alternative: Rename the Database in a Server Backup

1. **Export the database using mysqldump → mysqldump -u root -p nredb > nredb.sql**
2. **Create a new database CREATE DATABASE newdb;**
3. **Import the dump file into newdb → mysql -u root -p newdb < nredb.sql**
4. **Drop the old database → DROP DATABASE nredb;**

What is RID Organization (RID संस्था क्या है)?

- **RID Organization** यानि **Research, Innovation and Discovery Organization एक** संस्था हैं जो TWF (TWKSAA WELFARE FOUNDATION) **NGO** द्वारा RUN किया जाता है | जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही **इस RID Organization** की स्थपना 30.09.2023 किया गया है | जो TWF द्वारा संचालित किया जाता है |
- TWF (TWKSAA WELFARE FOUNDATION) NGO की स्थपना 26-10-2020 में बिहार की पावन धरती सासाराम में Er. RAJESH PRASAD एवं Er. SUNIL KUMAR द्वारा किया गया था जो की भारत सरकार द्वारा मान्यता प्राप्त संस्था हैं |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

MISSION, VISSION & MOTIVE OF "RID ORGANIZATION"	
मिशन	हर एक ONE भारत के संग
विजन	TALENT WORLD KA SHRESHTM AB AAYEGA भारत में और भारत का TALENT भारत में
मकसद	NEW (RID, PMS, TLR)

MOTIVE OF RID ORGANIZATION NEW (RID, PMS, TLR)		
NEW (RID)		
R	I	D
Research	Innovation	Discovery
NEW (TLR)		
T	L	R
Technology, Theory, Technique	Law	Rule
NEW (PMS)		
P	M	S
Product, Project, Production	Machine	Service

RID रीड संस्था की मिशन, विजन एवं मकसद को सार्थक हमें बनाना हैं |
भारत के वर्चस्व को हर कोने में फैलना हैं |
कर के नया कार्य एक बदलाव समाज में लाना हैं |
रीड संस्था की कार्य-सिध्दांतों से ही, हमें अपनी पहचान बनाना हैं |

Er. Rajesh Prasad (B.E, M.E)
Founder:
TWF & RID Organization

RESEARCH INNOVATION DISCOVERY
RID
BHARAT