### 1. ****SHOW DATABASES****:

This command lists all the databases present in the MySQL instance.

SHOW DATABASES;

**Result**: You’ll see a list of databases, such as:

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mydb |

| test |

+--------------------+

### 2. ****CREATE DATABASE**** database\_name:

Creates a new database.

CREATE DATABASE student\_db;

**Result**: A new database named student\_db is created.

### 3. ****DROP DATABASE**** database\_name:

Deletes a specified database permanently.

DROP DATABASE student\_db;

**Result**: The student\_db database is deleted from the server.

### 4. ****USE**** database\_name:

Selects a database to work with.

USE student\_db;

**Result**: All subsequent queries will be executed on student\_db.

### 5. ****SELECT DATABASE()****:

Displays the currently selected database.

SELECT DATABASE();

**Result**: If you are using student\_db, it will return:

+--------------+

| DATABASE() |

+--------------+

| student\_db |

+--------------+

### 6. ****CREATE TABLE**** table\_name:

Creates a table in the selected database.

CREATE TABLE students (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50),

age INT,

course VARCHAR(50)

);

**Result**: A students table with three columns (id, name, age, and course) is created.

### 7. ****SHOW TABLES****:

Lists all tables in the current database.

SHOW TABLES;

**Result**:

+--------------------+

| Tables\_in\_student\_db|

+--------------------+

| students |

+--------------------+

### 8. ****DESC**** table\_name:

Describes the structure of a table (i.e., its columns and data types).

DESC students;

**Result**:

+---------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------+-------------+------+-----+---------+----------------+

| id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(50) | YES | | NULL | |

| age | int | YES | | NULL | |

| course | varchar(50) | YES | | NULL | |

+---------+-------------+------+-----+---------+----------------+

### 9. ****SHOW COLUMNS FROM**** table\_name:

Similar to DESC, it shows all the columns and their data types.

SHOW COLUMNS FROM students;

**Result**: Same output as DESC students.

### 10. ****INSERT INTO**** table\_name ****(columns) VALUES (values)****:

Inserts data into the specified columns.

INSERT INTO students (name, age, course)

VALUES ('Amarjeet', 25, 'Java Full Stack');

**Result**: A new row is added to the students table.

### 11. ****SELECT \* FROM**** table\_name:

Retrieves all rows from a table.

SELECT \* FROM students;

**Result**:

+----+----------+-----+-------------------+

| id | name | age | course |

+----+----------+-----+-------------------+

| 1 | Amarjeet | 25 | Java Full Stack |

+----+----------+-----+-------------------+

### 12. ****ALTER TABLE**** table\_name ****DROP**** column\_name:

Removes a column from the table.

ALTER TABLE students DROP course;

**Result**: The course column is removed.

### 13. ****ALTER TABLE**** table\_name ****ADD**** column\_namedata\_type:

Adds a new column to the table.

ALTER TABLE students ADD course VARCHAR(50);

**Result**: The course column is added again.

### 14. Add a column at the first position:

ALTER TABLE employees

ADD COLUMN address VARCHAR(255) FIRST;

This adds the address column at the first position in the employees table.

### 15. Add a column after a specific column:

ALTER TABLE employees

ADD COLUMN phone\_number VARCHAR(15) AFTER last\_name;

### 16. ****UPDATE**** table\_name ****SET**** column\_name ****= value****:

Updates all records in the table.

UPDATE students SET age = 26;

**Result**: All students will now have age = 26.

### 17. ****DELETE FROM**** table\_name:

Deletes all records from a table.

DELETE FROM students;

**Result**: The table is now empty, but the structure remains intact.

### 18. ****DROP TABLE**** table\_name:

Deletes a table from the database.

DROP TABLE students;

**Result**: The students table is removed.

### 19. ****ALTER TABLE old\_table\_name RENAME TO new\_table\_name****:

Renames a table.

ALTER TABLE students RENAME TO learners;

**Result**: The students table is now called learners.

### 20. ****SELECT FROM**** table\_name ****WHERE**** condition:

Retrieves records based on a condition.

SELECT \* FROM students WHERE name = 'Amarjeet';

**Result**: Retrieves only rows where the name is 'Amarjeet'.

### 21. ****UPDATE**** table\_name ****SET**** column\_name ****= value WHERE**** condition:

Updates specific rows based on a condition.

UPDATE students SET name = 'Amarjeet Kumar Singh' WHERE id = 1;

**Result**: The name is updated for the student with id = 1.

### 22. ****DELETE FROM**** table\_name ****WHERE**** condition:

Deletes specific rows from the table based on a condition.

DELETE FROM students WHERE name = 'Sona';

**Result**: Deletes rows where the name is 'Sona'.

### 23. ****SELECT \* FROM**** table\_name ****LIMIT**** range:

Retrieves a limited number of rows from the table.

SELECT \* FROM students LIMIT 5;

**Result**: Only the first 5 rows are retrieved.

SELECT \* FROM students LIMIT 1, 5;

**Result**: Skips the first row and retrieves the next 5 rows.

### 24. ****SELECT**** column\_name ****FROM**** table\_name ****LIMIT**** range:

Retrieves specific columns with a limit.

SELECT id, name FROM students LIMIT 1, 4;

**Result**: Retrieves id and name of the first 4 rows, skipping the first one.

### 25. ****SELECT \* FROM**** table\_name ****WHERE**** condition:

Another example of retrieving data based on a condition.

SELECT \* FROM students WHERE id = 4;

**Result**: Retrieves the row with id = 4.

SELECT \* FROM students WHERE name = 'Amarjeet Kumar Singh';

**Result**: Retrieves rows where name is 'Amarjeet Kumar Singh'.

SELECT \* FROM students WHERE course = 'Java Full Stack';

**Result**: Retrieves all students enrolled in the 'Java Full Stack' course.

### ****26. Updating Multiple Columns for All Rows****

You can update multiple columns for all rows in a table without any condition.

#### Syntax:

UPDATE table\_name

SET column1 = value1, column2 = value2;

#### Example:

UPDATE employees

SET salary = 50000, department = 'HR';

This example sets the salary to 50,000 and department to 'HR' for **all employees** in the employees table.

### ****27. Updating Multiple Records Based on a Condition****

You can update multiple rows based on a specific condition using the WHERE clause.

#### Syntax:

UPDATE table\_name

SET column1 = value1, column2 = value2

WHERE condition;

#### Example:

UPDATE employees

SET salary = 60000, department = 'Finance'

WHERE id = 101;

This example updates the salary and department for the employee with id = 101. Only this specific employee's record is changed.

### ****28. Updating Different Values for Different Rows Using**** CASE

You can use the CASE statement to conditionally update different records with different values in a single UPDATE query. This is useful when updating multiple rows with different values based on conditions.

#### Syntax:

UPDATE table\_name

SET column\_name = CASE

WHEN condition1 THEN value1

WHEN condition2 THEN value2

ELSE default\_value

END

WHERE condition;

#### Example:

UPDATE employees

SET salary = CASE

WHEN id = 101 THEN 70000

WHEN id = 102 THEN 80000

ELSE salary

END

WHERE id IN (101, 102);

This example updates the salary for employees with specific id values:

* id = 101: salary becomes 70,000.
* id = 102: salary becomes 80,000.
* Other employees’ salaries remain unchanged.

### ****29. Updating Multiple Rows with a List of Conditions Using**** CASE

To update different rows with different values based on specific conditions, you can use multiple WHEN clauses inside the CASE statement.

#### Syntax:

UPDATE table\_name

SET column\_name = CASE

WHEN condition1 THEN new\_value1

WHEN condition2 THEN new\_value2

END

WHERE condition;

#### Example:

UPDATE employees

SET department = CASE

WHEN id = 101 THEN 'HR'

WHEN id = 102 THEN 'Engineering'

WHEN id = 103 THEN 'Marketing'

END

WHERE id IN (101, 102, 103);

This example updates the department for multiple employees:

* Employee id = 101 is moved to the 'HR' department.
* Employee id = 102 is moved to 'Engineering'.
* Employee id = 103 is moved to 'Marketing'.

### 30. The DISTINCT keyword in SQL is used to remove duplicate rows from the result set of a SELECT query. It ensures that the query returns only unique values for the specified columns.

### SELECT DISTINCT department

### FROM employees;

### Or

### SELECT DISTINCT department, name

### FROM employees;

### Or

### SELECT DISTINCT department

### FROM employees

### WHERE name LIKE 'A%';

### 31. The ORDER BY keyword is used to sort the result-set in ascending or descending order.

### SELECT \* FROM Products ORDER BY Price;

### Or

### SELECT \* FROM Products ORDER BY Price DESC;

### Or

### SELECT \* FROM Products ORDER BY ProductName;

### Or

### SELECT \* FROM Products ORDER BY ProductName DESC;

### Or

### SELECT \* FROM Customers ORDER BY Country ASC, CustomerName DESC;

**SQL Commands we have done practical:**

 **SHOW**: A command used to display information about databases, tables, or other database objects.

 **DATABASES**: Refers to the collection of data organized in a structured way. Used in conjunction with the SHOW command to list all databases.

 **CREATE**: A command used to create a new database or table in the database management system.

 **DATABASE**: Refers to a specific collection of related data entries in a structured format.

 **DROP**: A command used to delete a database or table permanently from the database management system.

 **USE**: A command used to select a specific database to work with.

 **SELECT**: A command used to retrieve data from one or more tables in a database.

 **TABLE**: Refers to a collection of related data entries consisting of rows and columns within a database.

 **FROM**: A clause used in a SQL query to specify the table from which to retrieve data.

 **DESC**: A command used to display the structure of a table, showing column names and data types.

 **COLUMNS**: Refers to the vertical divisions in a table, each representing a specific attribute of the data.

 **INSERT INTO**: A command used to add new rows of data into a specified table.

 **VALUES**: A keyword used in conjunction with INSERT INTO to specify the actual data to be added.

 **ALTER**: A command used to modify the structure of an existing table, such as adding or dropping columns.

 **ADD**: A keyword used with the ALTER command to add new columns or constraints to an existing table.

 **SET**: A keyword used in the UPDATE statement to specify the column(s) to be modified and the new value(s).

 **DELETE**: A command used to remove existing records from a table.

 **WHERE**: A clause used to specify conditions that must be met for records to be selected or affected by the query.

 **UPDATE**: A command used to modify existing records in a table.

 **LIMIT**: A clause used to specify the maximum number of records to return in a query result.

 **RENAME**: A command used to change the name of an existing database or table.

 **PRIMARY KEY**: A column or a set of columns that uniquely identifies each row in a table.

 **AUTO\_INCREMENT**: A property that allows a column (usually a primary key) to automatically generate a unique value for new rows.

 **AFTER**: A keyword used in table modifications to specify the position of a new column relative to existing columns.

 **FIRST**: A keyword used to specify that a new column should be added at the beginning of a table.

 **DISTINCT**: A keyword used in a SELECT statement to remove duplicate rows from the result set.

 **ORDER BY**: A clause used to sort the result set of a query by one or more columns in ascending or descending order.