**DBMS LAB MANUAL**

**EX. No. 1 – DDL Commands using CREATE, ALTER, TRUNCATE, and DROP**

**AIM:**  
To perform DDL operations such as CREATE, ALTER, TRUNCATE, and DROP on a database table.

**1. CREATE TABLE**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...

);

**Example:**

CREATE TABLE Student (

RollNo INT,

Name VARCHAR(50),

Age INT

);

**2. ALTER TABLE**

**Syntax:**

ALTER TABLE table\_name ADD column\_name datatype;

**Example:**

ALTER TABLE Student ADD Email VARCHAR(100);

**3. TRUNCATE TABLE**

**Syntax:**

TRUNCATE TABLE table\_name;

**Example:**

TRUNCATE TABLE Student;

**4. DROP TABLE**

**Syntax:**

DROP TABLE table\_name;

**Example:**

DROP TABLE Student;

**Result:**  
The above DDL commands were executed successfully. The Student table was created, altered, truncated, and dropped without any errors.

**EX. No. 2 – DDL Commands with Constraints**

**AIM:**  
To understand and implement constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT, and NOT NULL in a table.

**1. PRIMARY KEY**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype PRIMARY KEY,

column2 datatype

);

**Example:**

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50)

);

**2. FOREIGN KEY**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

FOREIGN KEY (column2) REFERENCES referenced\_table(referenced\_column)

);

**Example:**

CREATE TABLE Employee (

EmpID INT PRIMARY KEY,

EmpName VARCHAR(50),

DeptID INT,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

**3. UNIQUE**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype UNIQUE

);

**Example:**

CREATE TABLE Users (

UserID INT PRIMARY KEY,

Email VARCHAR(100) UNIQUE

);

**4. CHECK**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype CHECK (condition)

);

**Example:**

CREATE TABLE Product (

ProductID INT PRIMARY KEY,

Price DECIMAL(10,2) CHECK (Price > 0)

);

**5. DEFAULT**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype DEFAULT default\_value

);

**Example:**

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

Status VARCHAR(20) DEFAULT 'Pending'

);

**6. NOT NULL**

**Syntax:**

CREATE TABLE table\_name (

column1 datatype NOT NULL

);

**Example:**

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(50) NOT NULL

);

**Result:**  
The table definitions with constraints were executed successfully. All constraints such as PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT, and NOT NULL were applied correctly to their respective columns.

**EX. No. 3 – DML Commands using INSERT and SELECT**

**AIM:**  
To perform data manipulation operations using INSERT to add records and SELECT to retrieve data from the table.

**1. INSERT**

**Syntax:**

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

**Example:**

INSERT INTO Department (DeptID, DeptName)

VALUES (1, 'Computer Science');

INSERT INTO Employee (EmpID, EmpName, DeptID)

VALUES (101, 'Alice', 1);

**2. SELECT (All Columns)**

**Syntax:**

SELECT \* FROM table\_name;

**Example:**

SELECT \* FROM Employee;

**3. SELECT (Specific Columns)**

**Syntax:**

SELECT column1, column2 FROM table\_name;

**Example:**

SELECT EmpName, DeptID FROM Employee;

**4. SELECT with WHERE Clause**

**Syntax:**

SELECT \* FROM table\_name

WHERE condition;

**Example:**

SELECT \* FROM Employee

WHERE DeptID = 1;

**Result:**  
The INSERT statements added data into the Department and Employee tables successfully. The SELECT statements retrieved data as expected, displaying correct rows and columns based on the queries.

**EX. No. 4 – DML Commands using UPDATE and DELETE**

**AIM:**  
To understand and apply UPDATE and DELETE commands for modifying and removing records in a table.

**1. UPDATE**

**Syntax:**

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

**Example:**

UPDATE Employee

SET EmpName = 'Alicia'

WHERE EmpID = 101;

**2. DELETE**

**Syntax:**

DELETE FROM table\_name

WHERE condition;

**Example:**

DELETE FROM Employee

WHERE EmpID = 101;

**3. DELETE ALL ROWS (Caution!)**

**Syntax:**

DELETE FROM table\_name;

**Example:**

DELETE FROM Employee;

**Result:**  
The UPDATE command modified the employee’s name successfully. The DELETE command removed the specified employee record correctly. All operations were executed without errors.

**EX. No. 5 – TCL Commands: COMMIT, SAVEPOINT, ROLLBACK**

**AIM:**  
To demonstrate the use of **Transaction Control Language (TCL)** commands — COMMIT, SAVEPOINT, and ROLLBACK — for managing transactions in SQL.

**1. COMMIT**

**Syntax:**

COMMIT;

**Example:**

INSERT INTO Department (DeptID, DeptName) VALUES (2, 'Mechanical');

COMMIT;

**2. SAVEPOINT**

**Syntax:**

SAVEPOINT savepoint\_name;

**Example:**

SAVEPOINT sp1;

INSERT INTO Department (DeptID, DeptName) VALUES (3, 'Electrical');

**3. ROLLBACK**

**Syntax:**

ROLLBACK TO savepoint\_name;

**Example:**

ROLLBACK TO sp1;

**4. FULL ROLLBACK**

**Syntax:**

ROLLBACK;

**Example:**

ROLLBACK;

**Result:**  
Transactions were controlled successfully.

* COMMIT saved the changes permanently.
* SAVEPOINT created a save point within the transaction.
* ROLLBACK reverted changes to a specific savepoint or completely, as required.

**EX. No. 6 – DCL Commands: GRANT and REVOKE**

**AIM:**  
To manage user privileges and access control using GRANT and REVOKE.

**1. GRANT**

**Syntax:**

GRANT privilege\_type ON object TO user;

**Example:**

GRANT SELECT, INSERT ON Department TO 'user1';

**Explanation:**  
This command grants SELECT and INSERT privileges on the Department table to the user user1.

**2. REVOKE**

**Syntax:**

REVOKE privilege\_type ON object FROM user;

**Example:**

REVOKE INSERT ON Department FROM 'user1';

**Explanation:**  
This command revokes the INSERT privilege on the Department table from the user user1.

**Result:**  
The privileges were granted and revoked successfully. The user user1 was able to perform the actions for which privileges were granted, and was restricted from actions where privileges were revoked.

**EX. No. 7 – SELECT with Various Clauses: WHERE, Pattern Matching**

**AIM:**  
To filter records using the WHERE clause and perform pattern matching using LIKE.

**1. WHERE Clause**

**Syntax:**

SELECT column1, column2 FROM table\_name WHERE condition;

**Example:**

SELECT EmpName FROM Employee WHERE DeptID = 1;

**Explanation:**  
This query retrieves the names of employees where the DeptID is 1.

**2. Pattern Matching (LIKE)**

**Syntax:**

SELECT column1, column2 FROM table\_name WHERE column LIKE pattern;

**Example:**

SELECT EmpName FROM Employee WHERE EmpName LIKE 'A%';

**Explanation:**  
This query selects employee names starting with the letter "A".

**Result:**  
The WHERE clause successfully filtered records based on the given condition. The LIKE operator performed pattern matching and retrieved names that matched the specified pattern.

**EX. No. 8 – SELECT with Various Clauses: BETWEEN, IN, Aggregate Function**

**AIM:**  
To retrieve records using the BETWEEN, IN clauses and perform aggregate functions.

**1. BETWEEN Clause**

**Syntax:**

SELECT column1, column2 FROM table\_name WHERE column BETWEEN value1 AND value2;

**Example:**

SELECT Price FROM Product WHERE Price BETWEEN 50 AND 100;

**Explanation:**  
This query selects products with a price between 50 and 100.

**2. IN Clause**

**Syntax:**

SELECT column1, column2 FROM table\_name WHERE column IN (value1, value2, ...);

**Example:**

SELECT EmpName FROM Employee WHERE DeptID IN (1, 2);

**Explanation:**  
This query retrieves employee names whose DeptID is either 1 or 2.

**3. Aggregate Function**

**Syntax:**

SELECT AGGREGATE\_FUNCTION(column) FROM table\_name;

**Example:**

SELECT AVG(Price) FROM Product;

**Explanation:**  
This query calculates the average price of products in the Product table.

**Result:**  
The BETWEEN and IN clauses were successfully used to filter records based on specified conditions. The aggregate function correctly calculated the average price of products.

**EX. No. 9 – SELECT with Various Clauses: GROUP BY, HAVING, ORDER BY**

**AIM:**  
To group data using GROUP BY, filter grouped data using HAVING, and order the result using ORDER BY.

**1. GROUP BY Clause**

**Syntax:**

SELECT column1, AGGREGATE\_FUNCTION(column2) FROM table\_name GROUP BY column1;

**Example:**

SELECT DeptID, COUNT(\*) FROM Employee GROUP BY DeptID;

**Explanation:**  
This query groups employees by their department ID and counts the number of employees in each department.

**2. HAVING Clause**

**Syntax:**

SELECT column1, AGGREGATE\_FUNCTION(column2) FROM table\_name GROUP BY column1 HAVING condition;

**Example:**

SELECT DeptID, COUNT(\*) FROM Employee GROUP BY DeptID HAVING COUNT(\*) > 1;

**Explanation:**  
This query groups employees by DeptID and counts the number of employees in each department, but only includes departments with more than 1 employee.

**3. ORDER BY Clause**

**Syntax:**

SELECT column1, column2 FROM table\_name ORDER BY column1 [ASC|DESC];

**Example:**

SELECT EmpName, Salary FROM Employee ORDER BY Salary DESC;

**Explanation:**  
This query retrieves employee names and salaries, ordered by salary in descending order.

**Result:**  
The GROUP BY clause successfully grouped the data. The HAVING clause filtered the grouped data, and the ORDER BY clause correctly sorted the result by salary in descending order.

**EX. No. 10 – Query with Subquery and Correlated Query**

**AIM:**  
To demonstrate the use of subqueries and correlated queries.

**1. Subquery**

**Syntax:**

SELECT column1, column2 FROM table\_name WHERE column1 IN (SELECT column1 FROM table\_name WHERE condition);

**Example:**

SELECT EmpName FROM Employee WHERE DeptID IN (SELECT DeptID FROM Department WHERE DeptName = 'Computer Science');

**Explanation:**  
This query retrieves the names of employees who belong to the Computer Science department.

**2. Correlated Subquery**

**Syntax:**

SELECT column1, column2 FROM table\_name t1 WHERE column1 = (SELECT column2 FROM table\_name t2 WHERE t1.column1 = t2.column1);

**Example:**

SELECT EmpName FROM Employee e WHERE EXISTS (SELECT 1 FROM Department d WHERE e.DeptID = d.DeptID AND d.DeptName = 'Mechanical');

**Explanation:**  
This query retrieves employee names from the Employee table where a related DeptID exists in the Department table with DeptName as 'Mechanical'.

**Result:**  
The **Subquery** successfully retrieved employee names from the Computer Science department, and the **Correlated Query** returned employee names working in the Mechanical department. The queries executed successfully with correct results.

**EX. No. 11 – Query with Joins: EquiJoin, InnerJoin, OuterJoin**

**AIM:**  
To perform different types of joins such as EquiJoin, InnerJoin, and OuterJoin to combine data from multiple tables.

**1. EquiJoin**

**Syntax:**

SELECT column1, column2 FROM table1, table2 WHERE table1.column = table2.column;

**Example:**

SELECT Employee.EmpName, Department.DeptName

FROM Employee, Department

WHERE Employee.DeptID = Department.DeptID;

**Explanation:**  
This query selects employee names and their corresponding department names using an EquiJoin between Employee and Department tables where the DeptID matches.

**2. Inner Join**

**Syntax:**

SELECT column1, column2 FROM table1 INNER JOIN table2 ON table1.column = table2.column;

**Example:**

SELECT Employee.EmpName, Department.DeptName

FROM Employee

INNER JOIN Department ON Employee.DeptID = Department.DeptID;

**Explanation:**  
This query selects employee names and their corresponding department names using an INNER JOIN between the Employee and Department tables.

**3. Outer Join (Left and Right)**

**Syntax:**

SELECT column1, column2 FROM table1 LEFT OUTER JOIN table2 ON table1.column = table2.column;

**Example:**

SELECT Employee.EmpName, Department.DeptName

FROM Employee

LEFT OUTER JOIN Department ON Employee.DeptID = Department.DeptID;

**Explanation:**  
This query retrieves employee names and their department names using a LEFT OUTER JOIN, showing all employees even if there is no corresponding department.

**Result:**  
All JOIN operations were executed successfully:

* The **EquiJoin** and **InnerJoin** returned matching records from both tables.
* The **OuterJoin** returned all records from the left table, even if there was no matching record in the right table.

**EX. No. 12 – Query with VIEW and INDEX**

**AIM:**  
To create and query data using VIEW and INDEX.

**1. VIEW**

**Syntax:**

CREATE VIEW view\_name AS SELECT column1, column2 FROM table\_name WHERE condition;

**Example:**

CREATE VIEW EmpDept AS

SELECT EmpName, DeptName

FROM Employee

INNER JOIN Department ON Employee.DeptID = Department.DeptID;

**Explanation:**  
This creates a VIEW named EmpDept that combines employee names and their respective department names.

**2. INDEX**

**Syntax:**

CREATE INDEX index\_name ON table\_name(column\_name);

**Example:**

CREATE INDEX idx\_deptname ON Employee(DeptID);

**Explanation:**  
This creates an index on the DeptID column of the Employee table to optimize search queries.

**Result:**

* The **VIEW** query successfully created a virtual table, combining employee names and department names.
* The **INDEX** query successfully created an index on DeptID, which improved query performance on the Employee table.

**EX. No. 13 – Query with AUTO\_INCREMENT SEQUENCES**

**AIM:**  
To generate an auto-incrementing sequence for a column.

**1. AUTO\_INCREMENT**

**Syntax:**

CREATE TABLE table\_name (

column\_name INT AUTO\_INCREMENT,

column2 datatype,

PRIMARY KEY (column\_name)

);

**Example:**

CREATE TABLE Employee (

EmpID INT AUTO\_INCREMENT,

EmpName VARCHAR(50),

DeptID INT,

PRIMARY KEY (EmpID)

);

**Explanation:**  
This query creates a table Employee with an AUTO\_INCREMENT on the EmpID column, so each new row will automatically generate a unique EmpID.

**Result:**  
The Employee table was created successfully with an auto-incrementing EmpID column. Each time a new record is inserted, the EmpID will automatically increment.

**EX. No. 14 – Simple Programming using REPEAT and WHILE**

**AIM:**  
To perform basic looping operations using REPEAT and WHILE statements.

**1. REPEAT Loop**

**Syntax:**

REPEAT

-- SQL statements;

UNTIL condition;

END REPEAT;

**Example:**

SET @counter = 1;

REPEAT

INSERT INTO Employee (EmpName, DeptID) VALUES ('Employee' + @counter, 1);

SET @counter = @counter + 1;

UNTIL @counter > 5;

END REPEAT;

**Explanation:**  
This loop inserts 5 employees with the name format 'Employee1', 'Employee2', etc., into the Employee table.

**2. WHILE Loop**

**Syntax:**

WHILE condition DO

-- SQL statements;

END WHILE;

**Example:**

SET @counter = 1;

WHILE @counter <= 5 DO

INSERT INTO Employee (EmpName, DeptID) VALUES ('Employee' + @counter, 2);

SET @counter = @counter + 1;

END WHILE;

**Explanation:**  
This loop also inserts 5 employees with the name format 'Employee1', 'Employee2', etc., but into a different department (DeptID = 2).

**Result:**  
The **REPEAT** and **WHILE** loops executed successfully, inserting 5 rows into the Employee table in both cases.

**EX. No. 15 – Simple Programming using CASE and LOOP**

**AIM:**  
To perform conditional operations using CASE and LOOP statements.

**1. CASE Statement**

**Syntax:**

SELECT column1,

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

ELSE result3

END

FROM table\_name;

**Example:**

SELECT EmpName,

CASE

WHEN Salary > 5000 THEN 'High'

WHEN Salary BETWEEN 3000 AND 5000 THEN 'Medium'

ELSE 'Low'

END AS Salary\_Level

FROM Employee;

**Explanation:**  
This query evaluates the salary of each employee and assigns them a level of 'High', 'Medium', or 'Low'.

**2. LOOP Statement**

**Syntax:**

LOOP

-- SQL statements;

END LOOP;

**Example:**

SET @counter = 1;

LOOP

INSERT INTO Employee (EmpName, DeptID) VALUES ('Employee' + @counter, 3);

SET @counter = @counter + 1;

IF @counter > 5 THEN

LEAVE;

END IF;

END LOOP;

**Explanation:**  
This loop inserts employees into the Employee table, with the EmpID incremented automatically, and stops after 5 iterations.

**Result:**

* The **CASE** statement worked as expected, categorizing salaries as 'High', 'Medium', or 'Low'.
* The **LOOP** statement successfully inserted 5 employees into the Employee table.

**EX. No. 16 – High-Level Programming Extensions – Procedures**

**AIM:**  
To create and use stored procedures for executing repetitive tasks.

**Stored Procedure**

**Syntax:**

CREATE PROCEDURE procedure\_name (parameters)

BEGIN

-- SQL statements;

END;

**Example:**

CREATE PROCEDURE InsertEmployee(IN emp\_name VARCHAR(50), IN dept\_id INT)

BEGIN

INSERT INTO Employee (EmpName, DeptID) VALUES (emp\_name, dept\_id);

END;

**Explanation:**  
This stored procedure InsertEmployee takes emp\_name and dept\_id as input parameters and inserts the data into the Employee table.

**Execution of Procedure:**

**Syntax:**

CALL procedure\_name(parameters);

**Example:**

CALL InsertEmployee('John Doe', 1);

**Explanation:**  
This executes the InsertEmployee procedure, inserting the name "John Doe" with DeptID 1 into the Employee table.

**Result:**  
The **Stored Procedure** executed successfully, inserting the record as expected.

**EX. No. 17 – High-Level Programming Extensions – Function**

**AIM:**  
To create and use functions that return values.

**Function**

**Syntax:**

CREATE FUNCTION function\_name (parameters)

RETURNS return\_type

BEGIN

-- SQL statements;

RETURN value;

END;

**Example:**

CREATE FUNCTION CalculateSalary(emp\_id INT)

RETURNS INT

BEGIN

DECLARE salary INT;

SELECT Salary INTO salary FROM Employee WHERE EmpID = emp\_id;

RETURN salary;

END;

**Explanation:**  
This function CalculateSalary takes emp\_id as input, retrieves the salary of the employee with that ID from the Employee table, and returns it.

**Execution of Function:**

**Syntax:**

SELECT function\_name(parameters);

**Example:**

SELECT CalculateSalary(1);

**Explanation:**  
This executes the CalculateSalary function, retrieving the salary of the employee with EmpID 1.

**Result:**  
The **Function** executed successfully, returning the correct salary value for the employee.

**EX. No. 18 – High-Level Programming Extensions – Cursors**

**AIM:**  
To use cursors for row-by-row processing of query results.

**Cursor Declaration**

**Syntax:**

DECLARE cursor\_name CURSOR FOR SELECT\_query;

**Example:**

DECLARE employee\_cursor CURSOR FOR SELECT EmpName FROM Employee;

**Explanation:**  
This cursor employee\_cursor is used to iterate over the EmpName column of the Employee table.

**Cursor Operation**

**Syntax:**

OPEN cursor\_name;

FETCH cursor\_name INTO variable\_name;

CLOSE cursor\_name;

**Example:**

OPEN employee\_cursor;

FETCH employee\_cursor INTO @emp\_name;

CLOSE employee\_cursor;

**Explanation:**  
This opens the employee\_cursor, fetches the employee names into the variable @emp\_name, and then closes the cursor.

**Result:**  
The **Cursor** executed successfully, allowing for row-by-row processing of employee names.

**EX. No. 19 – High-Level Programming Extensions – Triggers**

**AIM:**  
To create and use triggers that automatically perform actions in response to certain events.

**Trigger Creation**

**Syntax:**

CREATE TRIGGER trigger\_name

BEFORE/AFTER INSERT/UPDATE/DELETE ON table\_name

FOR EACH ROW

BEGIN

-- SQL statements;

END;

**Example:**

CREATE TRIGGER EmployeeInsert

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (Action, EmpID, Date)

VALUES ('INSERT', NEW.EmpID, NOW());

END;

**Explanation:**  
This trigger EmployeeInsert is activated **after** a new record is inserted into the Employee table and logs the action in the AuditLog table.

**Result:**  
The **Trigger** was created successfully and logs insert actions into the AuditLog table when a new employee is added.

**EX. No. 20 – MySQL String Functions – REPLACE, REPEAT, REVERSE, RIGHT, LEFT, RPAD, LPAD**

**AIM:**  
To use various string functions to manipulate data in MySQL.

**1. REPLACE**

**Syntax:**

SELECT REPLACE(string, old\_substring, new\_substring);

**Example:**

SELECT REPLACE('Hello World', 'World', 'MySQL');

**Explanation:**  
This query replaces the word 'World' with 'MySQL' in the string 'Hello World'.

**2. REPEAT**

**Syntax:**

SELECT REPEAT(string, number\_of\_times);

**Example:**

SELECT REPEAT('Hello ', 3);

**Explanation:**  
This query repeats the string 'Hello ' three times, resulting in 'Hello Hello Hello '.

**3. REVERSE**

**Syntax:**

SELECT REVERSE(string);

**Example:**

SELECT REVERSE('MySQL');

**Explanation:**  
This query reverses the string 'MySQL', returning 'LQyM'.

**4. RIGHT**

**Syntax:**

SELECT RIGHT(string, number\_of\_characters);

**Example:**

SELECT RIGHT('Hello World', 5);

**Explanation:**  
This query returns the last 5 characters of the string 'Hello World', which is 'World'.

**5. LEFT**

**Syntax:**

SELECT LEFT(string, number\_of\_characters);

**Example:**

SELECT LEFT('Hello World', 5);

**Explanation:**  
This query returns the first 5 characters of the string 'Hello World', which is 'Hello'.

**6. RPAD**

**Syntax:**

SELECT RPAD(string, length, pad\_string);

**Example:**

SELECT RPAD('Hello', 10, '!');

**Explanation:**  
This query pads the string 'Hello' to a length of 10 by appending '!' characters, resulting in 'Hello!!!!!'.

**7. LPAD**

**Syntax:**

SELECT LPAD(string, length, pad\_string);

**Example:**

SELECT LPAD('Hello', 10, '!');

**Explanation:**  
This query pads the string 'Hello' to a length of 10 by prepending '!' characters, resulting in '!!!!!Hello'.

**Result:**  
All **String Functions** executed successfully:

* The **REPLACE** function replaced the specified substring.
* The **REPEAT** function repeated the string as expected.
* The **REVERSE** function reversed the string correctly.
* The **RIGHT** and **LEFT** functions returned the correct number of characters.
* The **RPAD** and **LPAD** functions padded the string as intended.

**EX. No. 21 – MySQL String Functions – SPACE, SUBSTR, UPPER, LOWER, TRIM, LENGTH**

**AIM:**  
To use various string functions for text manipulation and measurement.

**1. SPACE**

**Syntax:**

SELECT SPACE(number\_of\_spaces);

**Example:**

SELECT SPACE(5);

**Explanation:**  
This query returns a string containing 5 spaces.

**2. SUBSTR**

**Syntax:**

SELECT SUBSTR(string, start\_position, length);

**Example:**

SELECT SUBSTR('Hello World', 1, 5);

**Explanation:**  
This query returns the substring starting from position 1 and of length 5 from the string 'Hello World', which is 'Hello'.

**3. UPPER**

**Syntax:**

SELECT UPPER(string);

**Example:**

SELECT UPPER('hello world');

**Explanation:**  
This query converts the string 'hello world' to uppercase, returning 'HELLO WORLD'.

**4. LOWER**

**Syntax:**

SELECT LOWER(string);

**Example:**

SELECT LOWER('HELLO WORLD');

**Explanation:**  
This query converts the string 'HELLO WORLD' to lowercase, returning 'hello world'.

**5. TRIM**

**Syntax:**

SELECT TRIM(string);

**Example:**

SELECT TRIM(' Hello World ');

**Explanation:**  
This query removes the leading and trailing spaces from the string ' Hello World ', returning 'Hello World'.

**6. LENGTH**

**Syntax:**

SELECT LENGTH(string);

**Example:**

SELECT LENGTH('Hello World');

**Explanation:**  
This query returns the length of the string 'Hello World', which is 11 characters.

**Result:**  
All **String Functions** executed successfully:

* The **SPACE** function returned the correct number of spaces.
* The **SUBSTR** function extracted the correct substring.
* The **UPPER** and **LOWER** functions converted the text to the correct case.
* The **TRIM** function removed spaces as expected.
* The **LENGTH** function returned the correct character count.

**EX. No. 22 – Database Connectivity Using PHP and MySQL**

**AIM:**  
To establish a connection between PHP and MySQL database and perform basic operations.

**Connecting PHP with MySQL Database**

**Syntax:**

<?php

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = "database\_name";

// Create connection

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

} else {

echo "Connected successfully";

}

?>

**Example:**  
This code connects PHP to the database\_name on the local MySQL server.

**Basic Query in PHP**

**Syntax:**

<?php

$sql = "SELECT \* FROM table\_name";

$result = $conn->query($sql);

if ($result->num\_rows > 0) {

while($row = $result->fetch\_assoc()) {

echo "id: " . $row["id"]. " - Name: " . $row["name"]. "<br>";

}

} else {

echo "0 results";

}

?>

**Explanation:**  
This PHP script retrieves data from the table\_name and displays the id and name columns.

**Result:**  
Successfully connected to the database and displayed the data from the table.

**EX. No. 23 – Case Scenario: MySQL Queries Implementation for Train Ticket Reservation System to Receive Tickets through Social Network**

**AIM:**  
To implement a MySQL query for a Train Ticket Reservation System integrated with social networks.

**Train Ticket Reservation System Query**

**Table Structure:**

CREATE TABLE TicketReservation (

TicketID INT AUTO\_INCREMENT PRIMARY KEY,

CustomerName VARCHAR(100),

TrainNumber VARCHAR(50),

DateOfJourney DATE,

SocialNetwork VARCHAR(100),

TicketStatus VARCHAR(50)

);

**Query to Insert Ticket Reservation:**

INSERT INTO TicketReservation (CustomerName, TrainNumber, DateOfJourney, SocialNetwork, TicketStatus)

VALUES ('John Doe', '101', '2025-05-01', 'Facebook', 'Booked');

**Explanation:**  
This query inserts a new train ticket reservation with customer details, social network integration, and ticket status.

**Result:**  
The reservation was successfully inserted with the correct values.

**EX. No. 24 – Case Scenario: MySQL Queries Implementation for College Admission Form**

**AIM:**  
To implement MySQL queries for a College Admission Form.

**College Admission Form Query**

**Table Structure:**

CREATE TABLE CollegeAdmission (

StudentID INT AUTO\_INCREMENT PRIMARY KEY,

StudentName VARCHAR(100),

Course VARCHAR(100),

AdmissionDate DATE,

Status VARCHAR(50)

);

**Query to Insert Admission Form:**

INSERT INTO CollegeAdmission (StudentName, Course, AdmissionDate, Status)

VALUES ('Alice Smith', 'Computer Science', '2025-05-10', 'Admitted');

**Explanation:**  
This query inserts a new student admission record into the CollegeAdmission table.

**Result:**  
The college admission form data was inserted successfully into the database.

**EX. No. 25 – Case Scenario: MySQL Queries Implementation for QR Enabled Automatic Bus Ticket Booking System**

**AIM:**  
To implement MySQL queries for a QR Enabled Automatic Bus Ticket Booking System.

**Bus Ticket Booking System Query**

**Table Structure:**

CREATE TABLE BusTicketBooking (

BookingID INT AUTO\_INCREMENT PRIMARY KEY,

PassengerName VARCHAR(100),

BusNumber VARCHAR(50),

TravelDate DATE,

QRCode VARCHAR(255),

Status VARCHAR(50)

);

**Query to Insert Bus Ticket Booking:**

INSERT INTO BusTicketBooking (PassengerName, BusNumber, TravelDate, QRCode, Status)

VALUES ('Michael Johnson', 'B123', '2025-06-01', 'QR1234567890', 'Booked');

**Explanation:**  
This query inserts a new bus ticket booking record with the passenger's name, bus number, travel date, and a QR code.

**Result:**  
The bus ticket booking was successfully inserted with the provided information.