**Module 4 – Introduction to DBMS**

**1) Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address.**

Step 1:

CREATE DATABASE school\_db;

Step 2: USE school\_db;

Step 3:

CREATE TABLE students ( student\_id INT PRIMARY KEY AUTO\_INCREMENT, student\_name VARCHAR(100) NOT NULL,

age INT, class VARCHAR(20), address VARCHAR(255)

);

1. **Insert five records into the students table and retrieve all records using the SELECT statement.**

Step 1:

INSERT INTO students (student\_name, age, class, address)

VALUES

('Amit Sharma', 15, '10A', 'Delhi'),

('Priya Singh', 16, '11B', 'Mumbai'),

('Rahul Mehta', 14, '9C', 'Kolkata'),

('Sneha Patel', 17, '12A', 'Ahmedabad'),

('Karan Verma', 15, '10B', 'Chennai');

Step 2:

SELECT \* FROM students;

1. **Write SQL queries to retrieve specific columns (student\_name and age) from the students table.**

SELECT student\_name, age

FROM students;

**Example:**

SELECT Rahul Mehta, 14

FROM students;

1. **Write SQL queries to retrieve all students whose age is greater than 10.**

SELECT \*

FROM students

WHERE age > 10;

1. **Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).**

Step 1:

CREATE TABLE teachers ( teacher\_id INT PRIMARY KEY AUTO\_INCREMENT, teacher\_name VARCHAR(100) NOT NULL, subject VARCHAR(50) NOT NULL, email VARCHAR(100) UNIQUE

);

1. **Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.**

Step 1:

ALTER TABLE students

ADD teacher\_id INT;

Step 2:

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id);

1. **Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.**

CREATE TABLE courses ( course\_id INT PRIMARY KEY AUTO\_INCREMENT, course\_name VARCHAR(100) NOT NULL, course\_credits INT NOT NULL

);

1. **Use the CREATE command to create a database university\_db.**

CREATE DATABASE university\_db;

USE university\_db;

1. **Modify the courses table by adding a column course\_duration using the ALTER command.**

ALTER TABLE courses

ADD course\_duration VARCHAR(50);

1. **Drop the course\_credits column from the courses table.**

ALTER TABLE courses

DROP COLUMN course\_credits;

1. **Drop the teachers table from the school\_db database.**

USE school\_db;

DROP TABLE teachers;

1. **Drop the students table from the school\_db database and verify that the table has been removed.**

Step 1:

USE school\_db;

Step 2:

DROP TABLE students;

Step 3:

SHOW TABLES;

1. **Insert three records into the courses table using the INSERT command.**

INSERT INTO courses (course\_name, course\_duration)

VALUES

('Mathematics', '6 months'),

('Computer Science', '1 year'),

('Physics', '4 months');

1. **Update the course duration of a specific course using the UPDATE command.**

UPDATE courses

SET course\_duration = '2 years'

WHERE course\_name = 'Computer Science';

1. **Delete a course with a specific course\_id from the courses table using the DELETE command.**

DELETE FROM courses

WHERE course\_id = 3;

1. **Retrieve all courses from the courses table using the SELECT statement.**

SELECT \*

FROM courses;

1. **Sort the courses based on course\_duration in descending order using ORDER BY.**

SELECT \*

FROM courses

ORDER BY course\_duration DESC;

1. **Limit the results of the SELECT query to show only the top two courses using LIMIT.**

SELECT \*

FROM courses

LIMIT 2;

1. **Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.**

Step 1:

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1'; CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

Step 2:

GRANT SELECT ON school\_db.courses TO 'user1'@'localhost';

Step 3:

FLUSH PRIVILEGES;

1. **Revoke the INSERT permission from user1 and give it to user2.**

Step 1:

REVOKE INSERT ON school\_db.courses FROM 'user1'@'localhost';

Step 2:

GRANT INSERT ON school\_db.courses TO 'user2'@'localhost';

Step 3:

FLUSH PRIVILEGES;

1. **Insert a few rows into the courses table and use COMMIT to save the changes.**

Step 1:

START TRANSACTION;

Step 2:

INSERT INTO courses (course\_name, course\_duration)

VALUES

('Chemistry', '5 months'),

('Biology', '6 months'),

('English', '4 months');

Step 3:

COMMIT;

1. **Insert additional rows, then use ROLLBACK to undo the last insert operation.**

Step 1:

START TRANSACTION;

Step 2:

INSERT INTO courses (course\_name, course\_duration)

VALUES

('History', '3 months'),

('Geography', '4 months');

Step 3:

ROLLBACK;

1. **Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**

Step 1:

START TRANSACTION;

Step 2:

UPDATE courses

SET course\_duration = '7 months'

WHERE course\_name = 'Mathematics';

Step 3:

SAVEPOINT before\_update\_cs;

Step 4:

UPDATE courses

SET course\_duration = '3 years'

WHERE course\_name = 'Computer Science';

Step 5:

ROLLBACK TO SAVEPOINT before\_update\_cs;

Step 6:

COMMIT;

**25) Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.**

Step 1:

CREATE TABLE departments (

dept\_id INT PRIMARY KEY AUTO\_INCREMENT, dept\_name VARCHAR(100) NOT NULL

);

Step 2:

CREATE TABLE employees ( emp\_id INT PRIMARY KEY AUTO\_INCREMENT, emp\_name VARCHAR(100) NOT NULL,

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

Step 3:

-- Insert departments

INSERT INTO departments (dept\_name)

VALUES ('HR'), ('IT'), ('Finance');

-- Insert employees

INSERT INTO employees (emp\_name, dept\_id)

VALUES

('Amit', 1),

('Priya', 2),

('Rahul', 2),

('Sneha', 3);

Step 4:

SELECT e.emp\_name, d.dept\_name

FROM employees e

INNER JOIN departments d

ON e.dept\_id = d.dept\_id;

1. **Use a LEFT JOIN to show all departments, even those without employees.**

SELECT d.dept\_name, e.emp\_name

FROM departments d

LEFT JOIN employees e

ON d.dept\_id = e.dept\_id;

1. **Group employees by department and count the number of employees in each department using GROUP BY.**

SELECT d.dept\_name, COUNT(e.emp\_id) AS num\_employees

FROM departments d

LEFT JOIN employees e

ON d.dept\_id = e.dept\_id

GROUP BY d.dept\_name;

1. **Use the AVG aggregate function to find the average salary of employees in each department.**

Step 1:

ALTER TABLE employees

ADD salary DECIMAL(10,2);

Step 2:

UPDATE employees

SET salary = 50000 WHERE emp\_name = 'Amit';

UPDATE employees

SET salary = 60000 WHERE emp\_name = 'Priya';

UPDATE employees

SET salary = 65000 WHERE emp\_name = 'Rahul';

UPDATE employees

SET salary = 55000 WHERE emp\_name = 'Sneha';

Step 3:

SELECT d.dept\_name, AVG(e.salary) AS avg\_salary

FROM departments d

LEFT JOIN employees e

ON d.dept\_id = e.dept\_id

GROUP BY d.dept\_name;

1. **Write a stored procedure to retrieve all employees from the employees table based on department.**

DELIMITER //

CREATE PROCEDURE GetEmployeesByDept(IN deptName VARCHAR(100))

BEGIN

SELECT e.emp\_id, e.emp\_name, e.salary, d.dept\_name

FROM employees e

INNER JOIN departments d

ON e.dept\_id = d.dept\_id

WHERE d.dept\_name = deptName;

END //

DELIMITER ;

1. **Write a stored procedure that accepts course\_id as input and returns the course details.**

DELIMITER //

CREATE PROCEDURE GetCourseDetails(IN c\_id INT)

BEGIN

SELECT course\_id, course\_name, course\_duration

FROM courses

WHERE course\_id = c\_id;

END //

DELIMITER ;

1. **Create a view to show all employees along with their department names**

CREATE VIEW EmployeeDepartmentView AS

SELECT e.emp\_id, e.emp\_name, e.salary, d.dept\_name

FROM employees e

INNER JOIN departments d

ON e.dept\_id = d.dept\_id;

1. **Modify the view to exclude employees whose salaries are below $50,000.**

-- Original view (example)

CREATE VIEW employee\_view AS

SELECT employee\_id, employee\_name, salary

FROM employees;

-- Modified view with salary filter

CREATE OR REPLACE VIEW employee\_view AS

SELECT employee\_id, employee\_name, salary

FROM employees

WHERE salary >= 50000;

1. **Create a trigger to automatically log changes to the employees table when a new employee is added.**

Step 1:

CREATE TABLE employee\_log ( log\_id INT AUTO\_INCREMENT PRIMARY KEY, employee\_id INT, action VARCHAR(50), log\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Step 2:

DELIMITER $$

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_log (employee\_id, action)

VALUES (NEW.employee\_id, 'INSERTED');

END$$

DELIMITER ;

**34) Create a trigger to update the last\_modified timestamp whenever an employee record is updated.**

**We need to make sure the employees table has a last\_modified column**

ALTER TABLE employees

ADD COLUMN last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE

CURRENT\_TIMESTAMP;

**Create the trigger**

DELIMITER $$

CREATE TRIGGER before\_employee\_update

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END$$

DELIMITER ;