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)

);
```

2) 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;
```

3) 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;
```

4) Write SQL queries to retrieve all students whose age is greater than 10.

```
SELECT *

FROM students

WHERE age > 10;
```

5) 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
);
```

6) 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);
```

7) 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

);
```

8) Use the CREATE command to create a database university_db.

```
CREATE DATABASE university_db;

USE university_db;
```

9) Modify the courses table by adding a column course_duration using the ALTER command.

```
ALTER TABLE courses

ADD course_duration VARCHAR(50);
```

10) Drop the course_credits column from the courses table.

```
ALTER TABLE courses

DROP COLUMN course_credits;
```

11) Drop the teachers table from the school_db database.

```
USE school_db;

DROP TABLE teachers;
```

12) 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;
```

13) 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');
```

15) Update the course duration of a specific course using the UPDATE command.

```
UPDATE courses

SET course_duration = '2 years'

WHERE course_name = 'Computer Science';
```

16) Delete a course with a specific course_id from the courses table using the DELETE command.

```
DELETE FROM courses

WHERE course_id = 3;
```

17) Retrieve all courses from the courses table using the SELECT statement.

```
SELECT *

FROM courses;
```

18) Sort the courses based on course_duration in descending order using ORDER BY.

```
SELECT *

FROM courses

ORDER BY course_duration DESC;
```

19) Limit the results of the SELECT query to show only the top two courses using LIMIT.

```
SELECT *

FROM courses

LIMIT 2;
```

20) 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;
```

21) 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;
```

22) 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;
```

23) 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;
```

24) 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;
```

26) 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;
```

27) 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;
```

28) 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;
```

29) 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 ;
```

30) 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 ;
```

31) 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;
```

32) 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;
```

33) 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 ;
```