Module-4) Se - 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.

Ans:

```
-- Step 1: Create a new database named 'school_db'
CREATE DATABASE school_db;

-- Step 2: Use the newly created database
USE school_db;

-- Step 3: Create a table named 'students'
CREATE TABLE students (
student_id INT PRIMARY KEY,
student_name VARCHAR(100),
age INT,
class VARCHAR(20),
address VARCHAR(255)
);
```

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

Ans:

-----inserting data into student table:-

INSERT INTO students (student_id, student_name, age, class, address)

VALUES

(1, 'Aarav Mehta', 14, '8A', '123 MG Road, Mumbai'),

(2, 'Isha Patel', 13, '7B', '456 Nehru Nagar, Surat'),

(3, 'Rohan Verma', 15, '9C', '789 Tilak Street, Delhi'),

(4, 'Neha Sharma', 12, '6A', '321 Park Lane, Jaipur'),

(5, 'Kabir Joshi', 14, '8B', '654 Station Road, Pune');

-----displaying table:-

SELECT * FROM students;

3.Write SQL queries to retrieve specific columns (student_name and age) from the students table.

Ans:

------query for retrieve specific column:-

SELECT student_name, age

FROM students;

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

Ans:

---------retrieving data of students whose age is >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).

Ans:

```
CREATE TABLE teachers (
teacher_id INT PRIMARY KEY,
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.

Ans:

Step 1: Alter the students table to add teacher_id

```
ALTER TABLE students
ADD teacher_id INT;
```

Step 2: Add FOREIGN KEY constraint on teacher_id

```
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.

Ans:

```
CREATE TABLE courses (
course_id INT PRIMARY KEY,
course_name VARCHAR(100),
course_credits INT
```

);

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

Ans:

CREATE DATABASE university_db;

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

Ans:

ALTER TABLE courses

ADD course_duration VARCHAR(50);

10.Drop the course_credits column from the courses table.

Ans:

ALTER TABLE courses

DROP COLUMN course_credits;

11. Drop the teachers table from the school_db database.

Ans:

DROP TABLE school_db.teachers;

12.Drop the students table from the school_db database and verify that the table has been removed.

Ans:

- **Step 1: Drop the students table**

DROP TABLE school_db.students;

- **Step 2: Verify that the table has been removed**

SHOW TABLES FROM school_db;

13.Insert three records into the courses table using the INSERT command.

Ans:

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES
(101, 'Computer Science', '6 months'),
(102, 'Mathematics', '1 year'),
(103, 'Physics', '8 months');
```

14.Update the course duration of a specific course using the UPDATE command.

Ans:

```
    UPDATE courses
SET course_duration = '9 months'
WHERE course_id = 102;
```

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

Ans:

```
    DELETE FROM courses
WHERE course_id = 103;
```

16.Retrieve all courses from the courses table using the SELECT statement.

Ans:

```
    SELECT * FROM courses;
```

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

Ans:

```
SELECT * FROM courses
ORDER BY course_duration DESC;
```

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

Ans:

```
SELECT * FROM courses
LIMIT 2;
```

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

Ans:

✅ **Step 1: Create the Users**

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

✅ **Step 2: Grant SELECT Permission to user1 on courses Table**

```
GRANT SELECT ON school_db.courses TO 'user1'@'localhost';
```

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

Ans:

✅ **Step 1: Revoke INSERT from user1**

```
REVOKE INSERT ON school_db.courses FROM 'user1'@'localhost';
```

✅ **Step 2: Grant INSERT to user2**

```
GRANT INSERT ON school_db.courses TO 'user2'@'localhost';
```

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

Ans:

```
    -- Start transaction (optional, depends on environment)
START TRANSACTION;


-- Insert rows
INSERT INTO courses (course_id, course_name, course_duration)
VALUES
  (104, 'Chemistry', '7 months'),
  (105, 'English Literature', '6 months');


-- Commit the transaction
COMMIT;
```

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

Ans:

```
    -- Start a new transaction
START TRANSACTION;


-- Insert additional rows
INSERT INTO courses (course_id, course_name, course_duration)
VALUES
```

```
    (106, 'Biology', '8 months'),
    (107, 'History', '5 months');

-- Roll back the last insert operation
ROLLBACK;
```

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

Ans:

```
        -- Start a transaction
START TRANSACTION;

-- Optional: initial update (this will be retained)
UPDATE courses
SET course_duration = '10 months'
WHERE course_id = 104;

-- Create a savepoint
SAVEPOINT before_second_update;

-- Update that we may want to undo
UPDATE courses
SET course_duration = '12 months'
WHERE course_id = 105;
```

```
-- Roll back only the second update
ROLLBACK TO SAVEPOINT before_second_update;


-- Commit the remaining changes
COMMIT;
```

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

Ans:

✅ **Step 1: Create departments Table**

```
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(100)
);
```

✅ **Step 2: Create employees Table**

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

✅ **Step 3: Insert Sample Data**

```
-- Insert departments
INSERT INTO departments (dept_id, dept_name)
VALUES
    (1, 'Human Resources'),
    (2, 'Finance'),
    (3, 'Engineering');

-- Insert employees
INSERT INTO employees (emp_id, emp_name, dept_id)
VALUES
    (101, 'Alice', 1),
    (102, 'Bob', 2),
    (103, 'Charlie', 3),
    (104, 'David', 3);
```

✅ **Step 4: Perform INNER JOIN**

```
SELECT
    employees.emp_id,
    employees.emp_name,
    departments.dept_name
FROM
    employees
INNER JOIN
    departments
```

```
ON
   employees.dept_id = departments.dept_id;
```

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

Ans:

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

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

Ans:

```
      SELECT
   departments.dept_name,
   COUNT(employees.emp_id) AS employee_count
FROM
   departments
```

```
LEFT JOIN
   employees
ON
   departments.dept_id = employees.dept_id
GROUP BY
   departments.dept_name;
```

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

Ans:

✅ **Step 1: Add a salary column (if not already present)**

```
ALTER TABLE employees
ADD salary DECIMAL(10, 2);
```

✅ **Step 2: Update some salaries for demonstration**

```
UPDATE employees SET salary = 50000 WHERE emp_id = 101;
UPDATE employees SET salary = 60000 WHERE emp_id = 102;
UPDATE employees SET salary = 75000 WHERE emp_id = 103;
UPDATE employees SET salary = 80000 WHERE emp_id = 104;
```

✅ **Step 3: Use AVG() with GROUP BY to get average salary by department**

```
SELECT
   departments.dept_name,
   AVG(employees.salary) AS average_salary
```

```
FROM
   departments
LEFT JOIN
   employees ON departments.dept_id = employees.dept_id
GROUP BY
   departments.dept_name;
```

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

Ans:

```
        DELIMITER //


CREATE PROCEDURE get_employees_by_department(IN deptName
VARCHAR(100))
BEGIN
   SELECT
      employees.emp_id,
      employees.emp_name,
      employees.salary,
      departments.dept_name
   FROM
      employees
   INNER JOIN
      departments ON employees.dept_id = departments.dept_id
   WHERE
```

```
    departments.dept_name = deptName;
END //

DELIMITER ;
```

29.Write a stored procedure that accepts course_id as input and returns the course details.

Ans:

```
----------creating course table:-
courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100),
   course_credits INT
)
-----------stored procedure:-
DELIMITER $$

CREATE PROCEDURE GetCourseDetails(IN input_course_id INT)
BEGIN
   SELECT *
   FROM courses
   WHERE course_id = input_course_id;
END $$

DELIMITER ;
```

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

Ans:

--------creating employee table:-

```
employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(100),
   department_id INT
)
```

----------creating department table:-

```
departments (
   department_id INT PRIMARY KEY,
   department_name VARCHAR(100)
)
```

----------creating view:-

```
CREATE VIEW employee_department_view AS
SELECT
   e.employee_id,
   e.employee_name,
   d.department_name
FROM
   employees e
INNER JOIN
   departments d ON e.department_id = d.department_id;
```

----------How to use view:-

SELECT * FROM employee_department_view;

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

Ans:

CREATE OR REPLACE VIEW employee_department_view AS

SELECT

e.employee_id,

e.employee_name,

e.salary,

d.department_name

FROM

employees e

JOIN

departments d ON e.department_id = d.department_id

WHERE

e.salary >= 50000;

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

Ans:

✅ **Step 1: Create a log table (if it doesn't already exist)**

CREATE TABLE employee_log (

log_id INT PRIMARY KEY AUTO_INCREMENT,

employee_id INT,

```
    employee_name VARCHAR(100),
    action VARCHAR(50),
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

✅ **Step 2: Create the trigger**

✅ **For MySQL / PostgreSQL:**

```
CREATE TRIGGER log_new_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (employee_id, employee_name, action)
    VALUES (NEW.employee_id, NEW.employee_name, 'INSERT');
END;
```

33.Create a trigger to update the last_modified timestamp whenever an employee record is updated.

Ans:

✅ **Step 1: Ensure the employees table has a last_modified column**

```
ALTER TABLE employees
ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
```

✅ **Step 2: Create the trigger**

✅ **For MySQL:**

```
CREATE TRIGGER update_last_modified
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   SET NEW.last_modified = CURRENT_TIMESTAMP;
END;
```

34.Write a PL/SQL block to print the total number of employees from the employees table.

Ans:

```
        DECLARE
   total_employees NUMBER;
BEGIN
   SELECT COUNT(*) INTO total_employees
   FROM employees;


   DBMS_OUTPUT.PUT_LINE('Total number of employees: ' ||
total_employees);
END;
/
```

35.Create a PL/SQL block that calculates the total sales from an orders table.

Ans:

```
        DECLARE
   total_sales NUMBER(10,2);
BEGIN
```

```
    SELECT SUM(order_amount) INTO total_sales
    FROM orders;

    DBMS_OUTPUT.PUT_LINE('Total Sales: $' || total_sales);
END;
/
```

36.Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

Ans:

```
        DECLARE
    emp_id       NUMBER := 101;  -- change this to the employee ID you want to check
    emp_dept     VARCHAR2(50);
BEGIN
    SELECT department_name INTO emp_dept
    FROM employees e
    JOIN departments d ON e.department_id = d.department_id
    WHERE e.employee_id = emp_id;

    IF emp_dept = 'Sales' THEN
       DBMS_OUTPUT.PUT_LINE('The employee works in the Sales department.');
    ELSIF emp_dept = 'HR' THEN
       DBMS_OUTPUT.PUT_LINE('The employee works in the HR department.');
```

```
    ELSE
        DBMS_OUTPUT.PUT_LINE('The employee works in another
department: ' || emp_dept);
    END IF;
END;
/
```

37.Use a FOR LOOP to iterate through employee records and display their names.

Ans:

```
        DECLARE
    CURSOR emp_cursor IS
        SELECT employee_name FROM employees;
BEGIN
    FOR emp_rec IN emp_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Employee Name: ' ||
emp_rec.employee_name);
    END LOOP;
END;
/
```

38.Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Ans:

```
        DECLARE
    -- Declare variables to hold employee details
    v_employee_id   employees.employee_id%TYPE;
```

```
    v_employee_name employees.employee_name%TYPE;
    v_salary       employees.salary%TYPE;

    -- Declare the explicit cursor
    CURSOR emp_cursor IS
        SELECT employee_id, employee_name, salary
        FROM employees;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;

    LOOP
        -- Fetch each record into variables
        FETCH emp_cursor INTO v_employee_id, v_employee_name,
v_salary;

        -- Exit when no more rows
        EXIT WHEN emp_cursor%NOTFOUND;

        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('ID: ' || v_employee_id || ', Name: '
|| v_employee_name || ', Salary: ' || v_salary);
    END LOOP;
```

```
    -- Close the cursor
    CLOSE emp_cursor;
END;
/
```

39.Create a cursor to retrieve all courses and display them one by one.

Ans:

```
        DECLARE
    -- Variables to hold course details
    v_course_id     courses.course_id%TYPE;
    v_course_name   courses.course_name%TYPE;
    v_course_credit courses.course_credits%TYPE;

    -- Declare the cursor
    CURSOR course_cursor IS
        SELECT course_id, course_name, course_credits
        FROM courses;
BEGIN
    -- Open the cursor
    OPEN course_cursor;

    LOOP
        -- Fetch each course into variables
```

```
        FETCH course_cursor INTO v_course_id, v_course_name,
v_course_credit;

        -- Exit loop when no more rows
        EXIT WHEN course_cursor%NOTFOUND;

        -- Display course details
        DBMS_OUTPUT.PUT_LINE('Course ID: ' || v_course_id ||
                             ', Name: ' || v_course_name ||
                             ', Credits: ' || v_course_credit);
    END LOOP;

    -- Close the cursor
    CLOSE course_cursor;
END;
/
```

40.Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

Ans:

```
        BEGIN
    -- Start of transaction

    -- First insert
    INSERT INTO employees (employee_id, employee_name, salary,
department_id)
```

```
    VALUES (201, 'Alice Johnson', 60000, 10);

    -- Create a savepoint after first insert
    SAVEPOINT after_first_insert;

    -- Second insert
    INSERT INTO employees (employee_id, employee_name, salary,
department_id)
    VALUES (202, 'Bob Smith', 55000, 20);

    -- Rollback to savepoint (undo Bob's insert, keep Alice's)
    ROLLBACK TO after_first_insert;

    -- Commit the transaction to finalize Alice's insert
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Transaction complete: Inserted Alice,
rolled back Bob.');
END;
/
```

41.Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans:

```
        BEGIN
    -- Insert 1st employee (part to commit)
```

```
INSERT INTO employees (employee_id, employee_name, salary, department_id)
VALUES (301, 'Ravi Kumar', 60000, 1);

-- Insert 2nd employee (part to commit)
INSERT INTO employees (employee_id, employee_name, salary, department_id)
VALUES (302, 'Anjali Shah', 58000, 2);

-- Create a savepoint after first two inserts
SAVEPOINT after_first_two;

-- Commit the changes up to the savepoint
COMMIT;

-- Insert 3rd employee (this will be rolled back)
INSERT INTO employees (employee_id, employee_name, salary, department_id)
VALUES (303, 'Deepak Mehta', 62000, 3);

-- Insert 4th employee (this will also be rolled back)
INSERT INTO employees (employee_id, employee_name, salary, department_id)
VALUES (304, 'Pooja Verma', 61000, 4);
```

```
    -- Now rollback the remaining uncommitted changes
    ROLLBACK;

    DBMS_OUTPUT.PUT_LINE('First two inserts committed, remaining
rolled back.');
END;
/
```