Module 4 – Introduction to DBMS

LAB EXERCISES

NAME: - Rahul Zapadiya

DBMS Assignment

• Lab 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.

Answer: -

```
CREATE DATABASE school_db;


USE school_db;


CREATE TABLE students (
    student_id INT PRIMARY KEY,
```

```
    student_name VARCHAR(100),
    age INT,
    class VARCHAR(20),
    address VARCHAR(255)
);
```

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

Answer: -

```
INSERT INTO students (student_id, student_name, age, class, address)
VALUES
(1, 'Rahul Sharma', 15, '10th', 'Delhi'),
(2, 'Anjali Verma', 14, '9th', 'Mumbai'),
(3, 'Vikram Singh', 16, '11th', 'Chennai'),
```

(4, 'Priya Mehra', 15, '10th', 'Kolkata'),

(5, 'Aman Yadav', 13, '8th', 'Bangalore');

SELECT * FROM students;

2. SQL Syntax

- Lab 1: Write SQL queries to retrieve specific columns (student_name and age) from the students table.

Answer: -

SELECT student_name, age FROM students;

- Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

Answer: -

```
SELECT * FROM students
WHERE age > 10;
```

3. SQL Constraints

• Lab 1: Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

Answer: -

```
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(100) NOT NULL,
    subject VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE
);
```

• Lab 2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.

Answer: -

```
ALTER TABLE students
ADD teacher_id INT;

ALTER TABLE students
ADD CONSTRAINT fk_teacher
FOREIGN KEY (teacher_id)
REFERENCES teachers(teacher_id);
```

4. Main SQL Commands and Sub-commands (DDL)

• Lab 1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

Answer:-

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

• Lab 2: Use the CREATE command to create a database university_db.

Answer:-

```
CREATE DATABASE university_db;
```

5. ALTER Command

• Lab 1: Modify the courses table by adding a column course_duration using the ALTER command.

Answer: -

```
ALTER TABLE courses
ADD course_duration VARCHAR(50);
```

• Lab 2: Drop the course_credits column from the courses table.

Answer:-

```
ALTER TABLE courses
DROP COLUMN course_credits;
```

6. DROP Command

• Lab 1: Drop the teachers table from the school_db database.

Answer:-

```
USE school_db;

DROP TABLE teachers;
```

• Lab 2: Drop the students table from the school_db database and verify that the table has been removed.

Answer:-

```
USE school_db;

DROP TABLE students;

-- Verify that the table has been
removed
SHOW TABLES;
```

7. Data Manipulation Language (DML)

• Lab 1: Insert three records into the courses table using the INSERT command.

Answer:-

```
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES
(1, 'Mathematics', '6 months'),
(2, 'Physics', '4 months'),
(3, 'Computer Science', '8 months');
```

• Lab 2: Update the course duration of a specific course using the UPDATE command.

Answer:-

```
UPDATE courses
SET course_duration = '5 months'
WHERE course_id = 2;
```

• Lab 3: Delete a course with a specific course_id from the courses table using the DELETE command.

Answer:-

```
DELETE FROM courses
WHERE course_id = 3;
```

8. Data Query Language (DQL)

• Lab 1: Retrieve all courses from the courses table using the SELECT statement.

Answer:-

```
SELECT * FROM courses;
```

• Lab 2: Sort the courses based on course_duration in descending order using ORDER BY.

Answer:-

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

• Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Anser:-

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

LIMIT 2;

9. Data Control Language (DCL)

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

Answer:-

```
-- Create users (with passwords)
CREATE USER 'user1'@'localhost'
IDENTIFIED BY 'password1';
CREATE USER 'user2'@'localhost'
IDENTIFIED BY 'password2';

-- Grant SELECT permission on courses
table to user1
GRANT SELECT ON
database_name.courses TO
'user1'@'localhost';
```

• Lab 2: Revoke the INSERT permission from user1 and give it to user2.

Answer:-

```
-- Revoke INSERT permission from user1
REVOKE INSERT ON database_name.courses FROM 'user1'@'localhost';

-- Grant INSERT permission to user2
GRANT INSERT ON database_name.courses TO 'user2'@'localhost';
```

10. Transaction Control Language (TCL)

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

Answer:-

```
-- Insert rows
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES
(101, 'Mathematics', '6 months'),
(102, 'Physics', '4 months');

-- Save changes permanently
COMMIT;
```

• Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

Answer:-

```
-- Insert more rows
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES
(103, 'Chemistry', '5 months'),
```

(104, 'Biology', '3 months');

-- Undo the last insert operation
ROLLBACK;

• Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.
Answer:-
-- Create savepoint before update
SAVEPOINT before_update;

-- Update course duration
UPDATE courses
SET course_duration = '7 months'
WHERE course_id = 101;

-- Roll back only to the savepoint (undo the update)

```
ROLLBACK TO before_update;
```

11. SQL Joins

• Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

Answer:-

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

-- 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)
);

-- Insert sample data into departments
INSERT INTO departments (dept_id,
dept_name)
VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance');

-- Insert sample data into employees
INSERT INTO employees (emp_id,
emp_name, dept_id)
VALUES
(101, 'Alice', 1),
(102, 'Bob', 2),
```

(103, 'Charlie', 2);

-- INNER JOIN: show only employees who belong to a department
SELECT employees.emp_name, departments.dept_name
FROM employees
INNER JOIN departments
ON employees.dept_id = departments.dept_id;

• Lab 2: Use a LEFT JOIN to show all departments, even those without employees.
Answer:-
SELECT departments.dept_name, employees.emp_name
FROM departments
LEFT JOIN employees

ON departments.dept_id = employees.dept_id;

12. SQL Group By

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

Answer:-

```
-- Count employees in each department
SELECT dept_id, COUNT(emp_id) AS total_employees
FROM employees
GROUP BY dept_id;
```

• Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.

Answer:-

```
-- Find average salary per department
SELECT dept_id, AVG(salary) AS
avg_salary
FROM employees
GROUP BY dept_id;
```

13. SQL Stored Procedure

• Lab 1: Write a stored procedure to retrieve all employees from the employees table based

Answer:-

```
DELIMITER $$

CREATE PROCEDURE
GetEmployeesByDepartment(IN deptID
INT)
BEGIN
    SELECT *
```

```
    FROM employees
    WHERE dept_id = deptID;
END $$

DELIMITER ;
CALL GetCourseDetails(101);
```

• Lab 2: Write a stored procedure that accepts course_id as input and returns the course details.

Answer:-

```
DELIMITER $$

CREATE PROCEDURE
GetCourseDetails(IN c_id INT)
BEGIN
    SELECT *
    FROM courses
    WHERE course_id = c_id;
```

```
END $$

DELIMITER ;
CALL GetCourseDetails(101);
```

14. SQL View

• Lab 1: Create a view to show all employees along with their department names.

Answer:-

```
CREATE VIEW
EmployeeDepartmentView AS
SELECT e.emp_id, e.emp_name,
e.salary, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id =
d.dept_id;
```

```
SELECT * FROM
EmployeeDepartmentView;
```

• Lab 2: Modify the view to exclude employees whose salaries are below $50,000.

```
CREATE OR REPLACE VIEW
EmployeeDepartmentView AS
SELECT e.emp_id, e.emp_name,
e.salary, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id =
d.dept_id
WHERE e.salary >= 50000;

SELECT * FROM
EmployeeDepartmentView;
```

15. SQL Triggers

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

Answer:-

```
-- Create log table
CREATE TABLE employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    action VARCHAR(50),
    action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create trigger to log new employees
DELIMITER $$

CREATE TRIGGER log_new_employee
```

```
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, action)
    VALUES (NEW.emp_id, 'New employee added');
END $$

DELIMITER ;
```

• Lab 2: Create a trigger to update the last_modified timestamp whenever an employee record is updated.

Answer: -

```
DELIMITER $$

CREATE TRIGGER update_last_modified
```

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

DELIMITER ;
```

16. Introduction to PL/SQL

• Lab 1: Write a PL/SQL block to print the total number of employees from the employees table.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
    v_total_employees NUMBER;
BEGIN
```

```
    SELECT COUNT(*) INTO
v_total_employees
    FROM employees;

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

• Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
    v_total_sales NUMBER;
BEGIN
```

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

    DBMS_OUTPUT.PUT_LINE('Total
sales: $' || v_total_sales);
END;
/
```

17. PL/SQL Control Structures

• Lab 1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
```

```
    v_emp_id
employees.emp_id%TYPE := 101;  --
Change as needed
    v_dept_id
employees.dept_id%TYPE;
BEGIN
    SELECT dept_id
    INTO v_dept_id
    FROM employees
    WHERE emp_id = v_emp_id;

    IF v_dept_id = 1 THEN

DBMS_OUTPUT.PUT_LINE('Employee
belongs to HR department.');
    ELSIF v_dept_id = 2 THEN

DBMS_OUTPUT.PUT_LINE('Employee
belongs to IT department.');
```

```
    ELSE

DBMS_OUTPUT.PUT_LINE('Employee
belongs to another department.');
    END IF;
END;
/
```

• Lab 2: Use a FOR LOOP to iterate through employee records and display their names.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR emp_cursor IS
        SELECT emp_name FROM
employees;
BEGIN
```

```
    FOR emp_rec IN emp_cursor LOOP

DBMS_OUTPUT.PUT_LINE('Employee
Name: ' || emp_rec.emp_name);
    END LOOP;
END;
/
```

18. SQL Cursors

• Lab 1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
    -- Declare the cursor
    CURSOR emp_cursor IS
```

```
        SELECT emp_id, emp_name,
dept_id, salary
        FROM employees;

    -- Variables to hold each column
value
    v_emp_id
employees.emp_id%TYPE;
    v_emp_name
employees.emp_name%TYPE;
    v_dept_id
employees.dept_id%TYPE;
    v_salary   employees.salary%TYPE;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;

    LOOP
        -- Fetch the next row into variables
```

```
        FETCH emp_cursor INTO
v_emp_id, v_emp_name, v_dept_id,
v_salary;

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

        -- Display employee details
        DBMS_OUTPUT.PUT_LINE('ID: ' ||
v_emp_id ||
                             ', Name: ' ||
v_emp_name ||
                             ', Dept ID: ' ||
v_dept_id ||
                             ', Salary: ' || v_salary);
    END LOOP;

    -- Close the cursor
```

```
    CLOSE emp_cursor;
END;
/
```

• Lab 2: Create a cursor to retrieve all courses and display them one by one.

Answer: -

```
SET SERVEROUTPUT ON;

DECLARE
    -- Declare the cursor
    CURSOR course_cursor IS
        SELECT course_id, course_name, course_duration
        FROM courses;

    -- Variables to hold each column value
```

```
    v_course_id
courses.course_id%TYPE;
    v_course_name
courses.course_name%TYPE;
    v_course_duration
courses.course_duration%TYPE;
BEGIN
    -- Open the cursor
    OPEN course_cursor;

    LOOP
        -- Fetch each row into variables
        FETCH course_cursor INTO
v_course_id, v_course_name,
v_course_duration;

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

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

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

19. Rollback and Commit Savepoint

• Lab 1: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

Answer: -

```
-- Start the transaction
START TRANSACTION;

-- Insert first set of records
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES (201, 'Database Systems', '6
months');

-- Create a savepoint
SAVEPOINT sp1;

-- Insert second set of records
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES (202, 'Operating Systems', '5
months');
```

-- Roll back to savepoint (removes the second insert only)
ROLLBACK TO sp1;

-- Check current data
SELECT * FROM courses;

-- Commit remaining changes
COMMIT;

• Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.
Answer: -
-- Start the transaction
START TRANSACTION;

-- Insert first record

```
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES (301, 'Data Structures', '4
months');

-- Create a savepoint
SAVEPOINT sp2;

-- Insert second record
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES (302, 'Networks', '3 months');

-- Commit changes up to this point
(keeps both inserts so far)
RELEASE SAVEPOINT sp2;
COMMIT;

-- Insert third record
```

```
INSERT INTO courses (course_id,
course_name, course_duration)
VALUES (303, 'AI Basics', '2 months');

-- Decide to roll back the last insert
only
ROLLBACK;
```