Module 4 – Introduction to DBMS

LAB EXERCISES

1.Introduction to SQL

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.

Ans : CREATE DATABASE school_db;

USE school_db;

CREATE TABLE students(student_id INT PRIMARY KEY, student_name VARCHAR(100), age INT, class VARCHAR(10), address VARCHAR(50));

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

Ans :

1. INSERT INTO students(student_id, student_name, age, class, address)VALUES(1,'Meet Solanki',21,'A','Ahmedabad');
2. INSERT INTO students(student_id, student_name, age, class, address)VALUES(2,'Meet Gohel',21,'A','Jamnagar');

3. INSERT INTO students(student_id, student_name, age, class, address)VALUES(3,'Rahul Zapadiya',23,'B','Ahmedabad');
4. INSERT INTO students(student_id, student_name, age, class, address)VALUES(4,'Mahi Solanki',22,'C','Surat');
5. INSERT INTO students(student_id, student_name, age, class, address)VALUES(5,'Chirag Vaghela',20,'C','Rajkot');

SELECT * FROM students;

2.SQL Syntax

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

Ans : SELECT student_name, age FROM students;

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

Ans : 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).

Ans : CREATE TABLE teachers(teacher_id INT PRIMARY KEY, teacher_name VARCHAR(100) NOT NULL, subject VARCHAR(50) NOT NULL, email VARCHAR(150) UNIQUE);

Lab 2 : Implement a FOREGIN KEY constaint to relate the teacher_id from the teachers table with the students table.

Ans : ALTER TABLE students ADD teacher_id INT;

ALTER TABLE students ADD CONSTRAIN fk_teacher FOREIGN KEY(teacher_id) REFERENCES teacher(teacher_id);

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

Lab 1 : 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);

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

Ans : CREATE DATABASE university_db;

USE university_db;

5.ALTER Command

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

ALTER TABLE courses ADD course_duration VARCHAR(50);

Lab2 : Drop the course_credits column from the courses table.

Ans : ALTER TABLE courses DROP COLUMN course_credits;

6. DROP Command

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

Ans : USE school_db;

DROP TABLE IF EXISTS teachers;

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

Ans : DROP TABLE IF EXISTS students;

SHOW TABLE LIKE 'students';

7. Data Manipulation Language (DML)

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

Ans :

1.INSERT INTO courses (course_id, course_name, course_duration) VALUES (101, 'Computer Science Basics', '8 Months');

2. INSERT INTO courses (course_id, course_name, course_duration) VALUES (101, 'Computer Science Basics', '9 Months');

3. INSERT INTO courses (course_id, course_name, course_duration) VALUES (101, ‘’, '10 Months');

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

Ans: UPDATE courses

SET course_duration = '2.5 Months'

WHERE course_id = 102;

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

Ans: DELETE FROM courses

WHERE course_id = 103;

8. Data Query Language (DQL)

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

Ans : SELECT * FROM courses;

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

Ans : 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.

Ans : SELECT * FROM courses 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.

Ans : CREATE USER ‘user1’ IDENTIFIED BY ‘password1’;

CREATE USER ‘user2’ IDENTIFIED BY ‘password2’;

GRANT SELECT ON university_db.courses TO ‘user1’;

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

Ans : REVOKE INSERT ON university_db.courses FROM ‘user1’;

GRANT INSERT ON university_db.courses TO ‘user2’;

10. Transaction Control Language (TCL)

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

Ans : INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(201, 'Data Structures', '3 Months'),

(202, 'Operating Systems', '4 Months');

COMMIT;

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

Ans : INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(203, 'Machine Learning', '6 Months'),

(204, 'Artificial Intelligence', '6 Months');

ROLLBACK;

Lab 3 : Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans : SAVEPOINT before_update;

UPDATE courses

SET course_duration = '5 Months'

WHERE course_id = 201;

ROLLBACK TO before_update;

COMMIT;

11. SQL Joins

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

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

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

INSERT INTO departments (dept_id, dept_name) VALUES (1, 'HR');
INSERT INTO departments (dept_id, dept_name) VALUES (2, 'Finance');
INSERT INTO departments (dept_id, dept_name) VALUES (3, 'IT');
```

INSERT INTO departments (dept_id, dept_name) VALUES (4, 'Marketing');

INSERT INTO employees (emp_id, emp_name, dept_id) VALUES (101, 'Amit Sharma', 1);

INSERT INTO employees (emp_id, emp_name, dept_id) VALUES (102, 'Priya Mehta', 2);

INSERT INTO employees (emp_id, emp_name, dept_id) VALUES (103, 'Karan Patel', 3);

SELECT e.emp_name, d.dept_name FROM employees e INNER JOIN departments d ON e.dept_id = d.dept_id;

Lab 2 : Use a LEFT JOIN to show all departments, even those without employees.

Ans : SELECT d.dept_name, e.emp_name

FROM departments d

LEFT JOIN employees e

ON d.dept_id = e.dept_id;

12. SQL Group By

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

Ans : SELECT d.dept_name, COUNT(e.emp_id) AS total_employees FROM departments d LEFT JOIN employees e ON d.dept_id = e.dept_id GROUP BY d.dept_name;

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

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

UPDATE employees SET salary = 40000 WHERE emp_id = 101;

UPDATE employees SET salary = 50000 WHERE emp_id = 102;

UPDATE employees SET salary = 45000 WHERE emp_id = 103;

SELECT d.dept_name, AVG(e.salary) AS average_salary FROM departments d LEFT JOIN employees e ON d.dept_id = e.dept_id GROUP BY d.dept_name;

13. SQL Stored Procedure

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

Ans :DELIMITER ||

```
CREATE PROCEDURE GetEmployeesByDepartment(IN deptName VARCHAR(100))
BEGIN
   SELECT e.emp_id, e.emp_name, d.dept_name, e.salar FROM employees e INNER JOIN departments d ON e.dept_id = d.dept_id WHERE d.dept_name = deptName;
END||

DELIMITER;
```

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

Ans : DELIMITER||

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

DELIMITER ;
```

14. SQL View

Lab 1 : Create a view to showall employees along with their department names.

Ans : CREATE VIEW EmployeeDepartmentView AS

SELECT e.emp_id, e.emp_name, d.dept_name, e.salary FROM employees e INNER JOIN departments d ON e.dept_id = d.dept_id;

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

Ans : CREATE OR REPLACE VIEW EmployeeDepartmentView AS

SELECT e.emp_id, e.emp_name, d.dept_name, e.salary FROM employees e INNER JOIN departments d ON e.dept_id = d.dept_id WHERE e.salary >= 50000;

15. SQL Triggers

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

Ans : CREATE TABLE employee_log (log_id INT AUTO_INCREMENT PRIMARY KEY, emp_id INT, emp_name

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

DELIMITR ||

```
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employee_log (emp_id, emp_name, action)
   VALUES (NEW.emp_id, NEW.emp_name, 'INSERT');
END ||
```

DELIMITER ;

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

Ans : ALTER TABLE employees ADD last_modified TIMESTAMP NULL;

DELIMITER ||

```
CREATE TRIGGER before_employee_update
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.

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

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

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

Ans :

```
DECLARE
    total_sales NUMBER;
BEGIN
    SELECT SUM(order_amount) INTO total_sales
    FROM orders;

    DBMS_OUTPUT.PUT_LINE('Total Sales: ' || 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.

Ans :

```
DECLARE
    v_emp_name  employees.emp_name%TYPE;
```

```
    v_dept_name departments.dept_name%TYPE;
BEGIN
    -- Example: check for employee with emp_id = 101
    SELECT emp_name, dept_name
    INTO v_emp_name, v_dept_name
    FROM employees e
    INNER JOIN departments d
    ON e.dept_id = d.dept_id
    WHERE e.emp_id = 101;

    IF v_dept_name = 'IT' THEN
        DBMS_OUTPUT.PUT_LINE(v_emp_name || ' works in the
IT department.');
    ELSE
        DBMS_OUTPUT.PUT_LINE(v_emp_name || ' does not work
in the IT department.');
    END IF;
END;
```

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

Ans : 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.

Ans : DECLARE

CURSOR emp_cursor IS

SELECT emp_id, emp_name, salary

FROM employees;

v_emp_id employees.emp_id%TYPE;

v_emp_name employees.emp_name%TYPE;

```
    v_salary employees.salary%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_emp_id, v_emp_name, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('ID: ' || v_emp_id || ' | Name:
' || v_emp_name || ' | Salary: ' || v_salary);
    END LOOP;
    CLOSE emp_cursor;
END;
```

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

Ans :

```
DECLARE
    CURSOR course_cursor IS
        SELECT course_id, course_name, course_duration
        FROM courses;

    v_course_id courses.course_id%TYPE;
```

```
    v_course_name courses.course_name%TYPE;
    v_course_duration courses.course_duration%TYPE;
BEGIN
    OPEN course_cursor;
    LOOP
        FETCH course_cursor INTO v_course_id, v_course_name,
v_course_duration;
        EXIT WHEN course_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Course ID: ' || v_course_id ||
                    ' | Name: ' || v_course_name ||
                    ' | Duration: ' || v_course_duration);
    END LOOP;
    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.

Ans : START TRANSACTION;

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES (301, 'Cloud Computing', '3 Months');

SAVEPOINT before_extra_inserts;

INSERT INTO courses (course_id, course_name, course_duration)
VALUES
(302, 'Data Analytics', '4 Months'),
(303, 'Cyber Security', '5 Months');

ROLLBACK TO before_extra_inserts;

COMMIT;
```

Lab 2 : Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans : START TRANSACTION;

```
INSERT INTO courses (course_id, course_name, course_duration)
VALUES
```

(304, 'Artificial Intelligence', '6 Months'),

(305, 'Blockchain Technology', '4 Months');

SAVEPOINT first_batch;

RELEASE SAVEPOINT first_batch; -- (optional step to free the savepoint)

COMMIT; -- Commits everything done so far

START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration)

VALUES

(306, 'Internet of Things', '3 Months'),

(307, 'DevOps', '4 Months');

ROLLBACK;