Module 4 SQL LAB EXERCISE

LAB EXERCISES: 1

• 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.

```
CREATE DATABASE school_db;

CREATE TABLE students (

Std_id  INT NOT NULL  UNIQUE,

Name VARCHAR(15) NOT NULL,

Age INT NOT NULL,

Class VARCHAR(5)  NOT NULL,

Address  VARCHAR(30) NOT NULL

);
```

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

```
INSERT INTO students  VALUES (1,'RAVI ARYA',21,'10TH','PURSOTTAM SOCIETY PAVAN CITY MODASA');

INSERT INTO students  VALUES (2,'ROMIL RAJA',22,'11TH','PRUNIMA SOCIETY AHEMDABAD');

INSERT INTO students  VALUES (3,'AKHIL YADAV',21,'10TH','SOPAN CITY  RAJKOT');

INSERT INTO students  VALUES (4,'ANIL PANDIT',23,'12TH','REHNUMAN SOCIETY GANDHINAGAR');

INSERT INTO students  VALUES (5,'RADHIKA',23,'9TH','GURUKUL  SOCIETY ANAND');


SELECT *

FROM students;
```

Std_id	Name	Age	Class	Address
1	RAVI ARYA	10	10^{TH}	PURSOTTAM SOCIETY PAVAN CITY MODASA
2	ROMIL RAJA	12	11^{TH}	PRUNIMA SOCIETY AHEMDABAD
3	AKHIL YADAV	10	10^{TH}	SOPAN CITY RAJKOT
4	ANIL PANDIT	11	12^{TH}	REHNUMAN SOCIETY GANDHINAGAR
5	RADHIKA	13	9^{TH}	GURUKUL SOCIETY ANAND

LAB EXERCISES: 2

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

```
SELECT *

Name , age

FROM students
```

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

```
SELECT *

FROM student

WHERE age = 10;
```

LAB EXERCISES: 3

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

```
CREATE TABLE teachers (

Teacher_id INT PRIMARY KEY,

Teacher_name VARCHAR(20)  NOT NULL,

Subject VARCHAR(15) NOT NULL,

Email VARCHAR(25) NOT NULL UNIQUE

);
```

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

```
CREATE TABLE students (

   student_id INT PRIMARY KEY,

   student_name VARCHAR(100) NOT NULL,
```

```
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

Student Table Already create than I added column teacher_id and constraint FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)

```
INSERT INTO teachers(Teacher_name,Subject,Email)
VALUES ('Mr. Sharma', 'Maths', 'sharma@school.com');
INSERT INTO teachers(Teacher_name,Subject,Email)
VALUES ('Ms. Verma', 'Science', 'verma@school.com');
INSERT INTO teachers(Teacher_name,Subject,Email)
VALUES ('Mr. Pandit', 'Maths', 'Pandit@school.com');
INSERT INTO teachers(Teacher_name,Subject,Email)
VALUES ('Ms. Shyamnani', 'Science', 'Shyamnani@school.com');
```

LAB EXERCISES: 4

• Lab 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,
        course_name VARCHAR(20),
        course_credits float
 );
```

course_id	int	NO
course_name	varchar(20)	YES
course_duration	varchar(10)	YES

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

```
CREATE DATABASE university_db;
```

LAB EXERCISES: 5

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

course_id	int	NO	PRI
course_name	varchar(20)	YES	
course_credits	float	YES	
course_duration	varchar(10)	YES	

ALTER TABLE Courses ADD COLUMN course_duration VARCHAR(10);

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

ALTER TABLE Courses DROP COLUMN course_credits;

course_id	int	NO	PRI
course_name	varchar(20)	YES	
course_duration	varchar(10)	YES	

LAB EXERCISES: 6

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

DROP TABLE teachers;

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

0	49	17:04:44	desc students	Error Code: 1146. Table 'schooldb.students' doesn't exist	0.000 sec

LAB EXERCISES: 7

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

INSERT INTO Courses VALUES (101,'Maths','Two Year');

101	Maths	Two Year

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

```
UPDATE Courses SET course_duration = 'Five Year' WHERE course_id = 101;
```

101	Maths	Five Year

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

```
DELETE FROM Courses WHERE course_id =  101;
```

LAB EXERCISES: 8

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

Course_id	Subject	Course_duration
101	Maths	2
102	English	3
103	Physics	4
104	English	2

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

```
SELECT *

FROM Courses

WHERE course_duration

ORDER BY course_duration DESC;
```

Course_id	Subject	Course_duration
103	Physics	4
102	English	3
101	Maths	2
104	English	2

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

SELECT * FROM courses

LIMIT 2;

Course_id	Subject	Course_duration
101	Maths	2
102	English	3

LAB EXERCISES: 9

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

CREATE USER USER1 IDENTIFIED BY USER1@123 DEFAULT ROLE USER1;

GRANT SELECT ON universitydb.Courses TO USER1;

GRANT SELECT ON universitydb.Courses TO USER1;

REVOKE SELECT ON universitydb.Courses FROM USER1;

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

CREATE USER USER2 IDENTIFIED BY USER2@123 DEFAULT ROLE USER2;

GRANT SELECT ON universitydb.Courses TO USER2;

GRANT SELECT ON universitydb.Courses TO USER2;

LAB EXERCISES: 10

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

INSERT INTO Courses (course_id, course_name, course_credits) VALUES

(106, 'Python', 3.5),

(107, 'Java', 4.0),

(108, 'JavaScript', 3.0);

DELETE FROM courses WHERE course_id = 106;

DELETE FROM courses WHERE course_id = 107;

ROLLBACK;

COMMIT;

Course_id	Subject	Course_duration
101	C Language	3
102	C++	3.5
103	SQL	2.5
104	HTML/CSS	2
105	DSA	4
108	JavaScript	3

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

DELETE FROM courses WHERE course_id = 101;

DELETE FROM courses WHERE course_id = 102;

DELETE FROM courses WHERE course_id = 103;

Course_id	Subject	Course_duration
104	HTML/CSS	2
105	DSA	4

ROLLBACK;

Course_id	Subject	Course_duration
101	C Language	3
102	C++	3.5
103	SQL	2.5
104	HTML/CSS	2
105	DSA	4

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

```
SAVEPOINT S1;

DELETE FROM courses WHERE course_id = 101;


SAVEPOINT S2;

DELETE FROM courses WHERE course_id = 102;


SAVEPOINT S3;

DELETE FROM courses WHERE course_id = 103;

SELECT  * FROM Courses;
```

Course_id	Subject	Course_duration
101	C Language	3
102	C++	3.5
103	SQL	2.5
104	HTML/CSS	2
105	DSA	4
108	JavaScript	3

```
ROLLBACK TO S1;

SELECT * FROM courses;
```

Course_id	Subject	Course_duration
101	C Language	3
104	HTML/CSS	2
105	DSA	4
108	JavaScript	3

LAB EXERCISES: 11

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

```
CREATE TABLE departments (
dept_id INT PRIMARY KEY,
dept_name VARCHAR(50)
);

CREATE TABLE employees (
emp_id INT PRIMARY KEY,
emp_name VARCHAR(50),
salary FLOAT,
dept_id INT,
FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);

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;
```

Emp_id	Emp_name	Salary	Dept_name
102	Priya	30000	HR
101	Ravi	35000	IT
105	Raj	45000	IT
103	Amit	40000	Finance
104	Neha	32000	Marketing

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

Dept_id	Dept_name	Emp_id	Emp_name	Salary	Dept_id
1	HR	NULL	NULL	NULL	NULL
2	IT	NULL	NULL	NULL	NULL
3	Finance	NULL	NULL	NULL	NULL
4	Marketing	NULL	NULL	NULL	NULL

LAB EXERCISES: 12

• Lab 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 employee_count
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.dept_id
GROUP BY d.dept_name;
```

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

salary_of_all_employees_AVERAGE
36400

LAB EXERCISES: 13

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

```
DELIMITER //

CREATE PROCEDURE GetAllData()

BEGIN

SELECT *  FROM employees;

END //

DELIMITER  ;

CALL  GetAllData()
```

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

```
DELIMITER //

CREATE PROCEDURE getalldata(IN course_id INT)

BEGIN

SELECT * FROM courses WHERE course_id = course_id;

END //

DELIMITER ;

CALL  getalldata(1);
```

LAB EXERCISES: 14

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

```
CREATE TABLE departments (

   dept_id INT PRIMARY KEY,

   dept_name VARCHAR(50)

);
```

```
INSERT INTO departments VALUES
 (1, 'HR'), (2, 'Sales'), (3, 'IT');
--------------------------------------------------
CREATE TABLE employees (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50),
   salary FLOAT,
   dept_id INT,
   FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
INSERT INTO employees VALUES
(101, 'Ravi', 60000, 1),
(102, 'Priya', 45000, 2),
(103, 'Amit', 70000, 1),
(104, 'Neha', 50000, 3);
CREATE VIEW employee_with_department 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 employee_with_department;
```

Emp_id	Emp_name	Salary	Dept_name
101	Ravi	60000	HR
102	Priya	45000	Sales
103	Amit	70000	HR
104	Neha	50000	IT

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

```
CREATE OR REPLACE VIEW employee_with_department 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 employee_with_department;
```

Emp_id	Emp_name	Salary	Dept_name
101	Ravi	60000	HR
103	Amit	70000	HR
104	Neha	50000	IT

LAB EXERCISES: 15

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

```
DELIMITER //

CREATE TRIGGER after_employee_insert

AFTER INSERT ON Main_Employees

FOR EACH ROW

BEGIN

   INSERT INTO employee_log(emp_id, name, action)

   VALUES (NEW.emp_id, NEW.name, 'INSERT');

END //

DELIMITER ;

INSERT INTO Main_Employees (emp_id, name, department, salary)

VALUES (101, 'Ravi', 'IT', 40000);


SELECT * FROM employee_log;
```

Emp_id	Name	Stamp
101	Ravi	2025-07-09 17:42:00

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

```
DELIMITER //

CREATE TRIGGER before_employee_update

BEFORE UPDATE ON employees

FOR EACH ROW
```

```
BEGIN
   SET NEW.last_modified = CURRENT_TIMESTAMP;
END //
DELIMITER ;

-- Insert sample data
INSERT INTO employees (emp_id, name, department, salary)
VALUES (102, 'Amit', 'HR', 35000);

-- Update salary
UPDATE employees
SET salary = 40000
WHERE emp_id = 102;

-- Check updated timestamp
SELECT * FROM Main_Employees;
```

emp_id	name	department	salary	last_modified
102	Amit	HR	40000	2025-07-09 17:42:00

LAB EXERCISES: 16

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

```
DELIMITER $$

CREATE PROCEDURE show_employees()
```

```
BEGIN
  DECLARE DONE INT DEFAULT FALSE;

  -- Declare variables
  DECLARE e_id INT;
  DECLARE e_first_name VARCHAR(25);
  DECLARE e_last_name VARCHAR(25);
  DECLARE e_bob DATE;
  DECLARE e_salary INT;
  DECLARE e_comission_pct FLOAT;
  DECLARE e_department_id INT;
  DECLARE e_manager_id INT;

  -- Declare cursor
  DECLARE employees_cursor CURSOR FOR
    SELECT emp_id, first_name, last_name, bob, salary, e_comission_pct, department_id, manager_id
    FROM employees;

  -- Declare continue handler
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET DONE = TRUE;

  -- Open cursor
  OPEN employees_cursor;

  -- Read loop
```

```
  read_loop: LOOP
    FETCH employees_cursor INTO
      e_id, e_first_name, e_last_name, e_bob,
      e_salary, e_comission_pct, e_department_id, e_manager_id;

    IF DONE THEN
      LEAVE read_loop;
    END IF;

    -- Print each row using SELECT
    SELECT
      e_id AS ID,
      e_first_name AS FIRST_NAME,
      e_last_name AS LAST_NAME,
      e_bob AS DOB,
      e_salary AS SALARY,
      e_comission_pct AS COMMISSION,
      e_department_id AS DEPT_ID,
      e_manager_id AS MANAGER_ID;

  END LOOP;

  CLOSE employees_cursor;
END$$
```

```
DELIMITER ;

CALL show_employees();
```

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

```
DELIMITER $$

CREATE PROCEDURE show_courses()
BEGIN
  DECLARE done INT DEFAULT FALSE;

  -- Declare variables
  DECLARE c_id INT;
  DECLARE c_name VARCHAR(20);
  DECLARE c_credits FLOAT;

  -- Declare cursor
  DECLARE course_cursor CURSOR FOR
    SELECT course_id, course_name, course_credits FROM courses;

  -- Continue handler
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  -- Open cursor
```

```
  OPEN course_cursor;

  read_loop: LOOP
    FETCH course_cursor INTO c_id, c_name, c_credits;

    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Show each course record
    SELECT
      c_id AS Course_ID,
      c_name AS Course_Name,
      c_credits AS Credits;

  END LOOP;

  CLOSE course_cursor;
END$$

DELIMITER ;
call  show_courses();
```

LAB EXERCISES: 17

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

```
CREATE TABLE EMPLOMENT (
ID INT PRIMARY KEY,
NAME VARCHAR(15)
);
INSERT INTO EMPLOMENT (ID,NAME) VALUES (1,'RAVI');
SAVEPOINT SAVE1;

INSERT INTO EMPLOMENT (ID,NAME) VALUES (2,'MOHIT');
INSERT INTO EMPLOMENT (ID,NAME) VALUES (3,'DHRUV');

SELECT * FROM EMPLOMENT;
ROLLBACK TO SAVE1;
```

ID	NAME
1	RAVI

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

```
INSERT INTO EMPLOMENT (ID,NAME) VALUES (4,'RAMESH');
INSERT INTO EMPLOMENT (ID,NAME) VALUES (5,'SURESH');
SAVEPOINT SAVE2;

INSERT INTO EMPLOMENT (ID,NAME) VALUES (6,'MEENA');
INSERT INTO EMPLOMENT (ID,NAME) VALUES (7,'SEEMA');

COMMIT;
```

```
INSERT INTO EMPLOMENT (ID,NAME) VALUES (8,'TINA');

INSERT INTO EMPLOMENT (ID,NAME) VALUES (9,'MINA');


ROLLBACK;


SELECT * FROM EMPLOMENT;
```

ID	NAME
1	RAVI
4	RAMESH
5	SURESH
6	MEENA
7	SEEMA

----------THANK YOU----------