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(50) NOT NULL,

age INT,

class VARCHAR(20),

address VARCHAR(100)

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

Ans

INSERT INTO students (student\_id, student\_name, age, class, address)

VALUES

(1, 'jay kadiya', 18, '12th', 'Ahmedabad'),

(2, 'Ravi Kumar', 17, '11th', 'amreli'),

(3, 'Neha Patel', 16, '10th', 'Rajkot'),

(4, 'Karan Shah', 18, '12th', 'Vadodara'),

(5, 'Pooja Singh', 17, '11th', 'Gandhinagar');

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 age 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(50) NOT NULL,

subject VARCHAR(50) 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.

Ans

LTER 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.

Ans

CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

course\_credits INT

);

• Lab 2: Use the CREATE command to create a database university\_db.

Ans

CREATE DATABASE university\_db;

5. ALTER Command

Lab 1: Modify the courses table by adding a column course\_duration using the ALTER command.

Ans

ALTER TABLE coursesLab 1: Drop the teachers table from the school\_db database.ADD course\_duration VARCHAR(50);

Lab 2: 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 teachers;

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

Ans

USE school\_db;

DROP TABLE students;

SHOW TABLES;

7. Data Manipulation Language (DML)

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

Ans

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES

(1, 'Computer Science', '3 Years'),

(2, 'Business Administration', '2 Years'),

(3, 'Mathematics', '1 Year');

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

Ans

UPDATE courses

SET course\_duration = '2.5 Years'

WHERE course\_id = 2;

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 = 3;

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;

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'@'localhost' IDENTIFIED BY 'password1';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

GRANT SELECT ON school\_db.courses TO 'user1'@'localhost';

FLUSH PRIVILEGES;

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

Ans

REVOKE INSERT ON school\_db.courses FROM 'user1'@'localhost';

GRANT INSERT ON school\_db.courses TO 'user2'@'localhost';

FLUSH PRIVILEGES;

10. Transaction Control Language (TCL)

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

Ans

START TRANSACTION;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES

(4, 'Physics', '1.5 Years'),

(5, 'Chemistry', '2 Years'),

(6, 'English Literature', '3 Years');

COMMIT;

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

Ans

START TRANSACTION;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES

(7, 'Biology', '2 Years'),

(8, 'History', '1 Year');

ROLLBACK;

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

Ans

START TRANSACTION;

SAVEPOINT before\_update;

UPDATE courses

SET course\_duration = '4 Years'

WHERE course\_id = 1;

UPDATE courses

SET course\_duration = '3.5 Years'

WHERE course\_id = 2

ROLLBACK TO before\_update;

COMMIT;

11. SQL Joins

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

Ans

SELECT department, COUNT(\*) AS total\_employees

FROM employees

GROUP BY department;

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

Ans

SELECT department, AVG(salary) AS average\_salary

FROM employees

GROUP BY department;

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 dept\_name VARCHAR(100))

BEGIN

SELECT \*

FROM employees

WHERE department = dept\_name;

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 cid INT)

BEGIN

SELECT \*

FROM courses

WHERE course\_id = cid;

END $$

DELIMITER ;

14. SQL View

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

Ans

CREATE 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;

Lab 2: 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;

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 PRIMARY KEY AUTO\_INCREMENT,

employee\_id INT,

employee\_name VARCHAR(100),

action VARCHAR(50),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER after\_employee\_insert

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 $$

DELIMITER ;

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

Ans

ALTER TABLE employees

ADD COLUMN last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

ON UPDATE CURRENT\_TIMESTAMP;

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

SET SERVEROUTPUT ON;

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

SET SERVEROUTPUT ON;

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;

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

SET SERVEROUTPUT ON;

DECLARE

emp\_id employees.employee\_id%TYPE := 101; -- Example employee ID

emp\_dept employees.department%TYPE;

BEGIN

SELECT department INTO emp\_dept

FROM employees

WHERE employee\_id = emp\_id;

IF emp\_dept = 'Sales' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ' || emp\_id || ' works in the Sales department.');

END IF;

END;

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

ANS

SET SERVEROUTPUT ON;

BEGIN

FOR emp\_rec IN (SELECT employee\_id, employee\_name FROM employees) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp\_rec.employee\_id ||

', Name: ' || emp\_rec.employee\_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

SET SERVEROUTPUT ON;

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_id, employee\_name, department, salary

FROM employees;

v\_emp\_id employees.employee\_id%TYPE;

v\_emp\_name employees.employee\_name%TYPE;

v\_dept employees.department%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_emp\_id, v\_emp\_name, v\_dept, v\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_emp\_id ||

', Name: ' || v\_emp\_name ||

', Dept: ' || v\_dept ||

', 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

SET SERVEROUTPUT ON;

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

BEGIN;

SAVEPOINT before\_insert;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (101, 'Mathematics', '3 months');

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (102, 'Physics', '4 months');

SAVEPOINT after\_insert;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (103, 'Chemistry', '2 months');

ROLLBACK TO SAVEPOINT before\_insert;

COMMIT;

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

ANS

BEGIN;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (201, 'Biology', '3 months');

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (202, 'Computer Science', '6 months');

SAVEPOINT after\_first\_inserts;

COMMIT; -- Now first 2 inserts are permanent

BEGIN;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (203, 'History', '4 months');

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (204, 'Economics', '5 months');

ROLLBACK;

BEGIN

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (201, 'Biology', '3 months');

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (202, 'Computer Science', '6 months');

SAVEPOINT after\_first\_inserts;

COMMIT;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (203, 'History', '4 months');

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (204, 'Economics', '5 months');

ROLLBACK;

END;