1. Database and Table Creation

Lab 1:

```
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 INTO students VALUES
(1, 'Alice', 12, '6A', '123 Oak St'),
(2, 'Bob', 11, '5B', '456 Maple St'),
(3, 'Charlie', 10, '5A', '789 Pine St'),
(4, 'David', 13, '7B', '321 Cedar St'),
(5, 'Eve', 14, '8A', '654 Birch St');

SELECT * FROM students;
```

---

2. SQL Syntax

Lab 1:

```
SELECT student_name, age FROM students;
```

Lab 2:

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

---

3. SQL Constraints

Lab 1:

```
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:

```
ALTER TABLE students
ADD teacher_id INT,
ADD CONSTRAINT fk_teacher
```

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

---

4. DDL Commands

Lab 1:

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

Lab 2:

```
CREATE DATABASE university_db;
```

---

5. ALTER Command

Lab 1:

```
ALTER TABLE courses ADD course_duration INT;
```

Lab 2:

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

---

6. DROP Command

Lab 1:

```
DROP TABLE teachers;
```

Lab 2:

```
DROP TABLE students;
```

---

7. DML Commands

Lab 1:

```
INSERT INTO courses (course_id, course_name, course_duration) VALUES
(101, 'Math', 6),
(102, 'Science', 5),
(103, 'History', 4);
```

Lab 2:

```
UPDATE courses SET course_duration = 8 WHERE course_id = 101;
```

Lab 3:

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

---

8. DQL Commands

Lab 1:

```
SELECT * FROM courses;
```

Lab 2:

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

Lab 3:

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

---

9. DCL Commands

Lab 1:

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

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

Lab 2:

```
REVOKE INSERT ON university_db.courses FROM 'user1'@'localhost';
GRANT INSERT ON university_db.courses TO 'user2'@'localhost';
```

---

10. TCL Commands

Lab 1:

```
START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration) VALUES
(104, 'English', 6),
(105, 'Art', 3);

COMMIT;
```

Lab 2:

```
START TRANSACTION;

INSERT INTO courses (course_id, course_name, course_duration) VALUES
(106, 'Music', 4);
```

```
ROLLBACK;
```

Lab 3:

```
START TRANSACTION;

SAVEPOINT before_update;

UPDATE courses SET course_duration = 7 WHERE course_id = 104;

ROLLBACK TO before_update;

COMMIT;
```

---

11. SQL Joins

Lab 1:

```
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)
);

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

Lab 2:

```
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:

```
SELECT dept_id, COUNT(*) AS employee_count
FROM employees
GROUP BY dept_id;
```

Lab 2:

```
SELECT dept_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY dept_id;
```

13. SQL Stored Procedure

Lab 1:

```
CREATE PROCEDURE GetEmployeesByDept(IN dept_id INT)
BEGIN
  SELECT * FROM employees WHERE department_id = dept_id;
END;
```

Lab 2:

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

---

14. SQL View

Lab 1:

```
CREATE VIEW emp_with_dept AS
SELECT e.*, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

Lab 2:

```
CREATE OR REPLACE VIEW emp_with_dept AS
SELECT e.*, 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 TRIGGER log_new_employee
AFTER INSERT ON employees
FOR EACH ROW
INSERT INTO employee_log (employee_id, action, action_time)
VALUES (NEW.employee_id, 'INSERT', NOW());
```

Lab 2:

```
CREATE TRIGGER update_last_modified
BEFORE UPDATE ON employees
FOR EACH ROW
SET NEW.last_modified = NOW();
```

---

16. Introduction to PL/SQL

Lab 1:

```
DECLARE
  emp_count INT;
BEGIN
  SELECT COUNT(*) INTO emp_count FROM employees;
  DBMS_OUTPUT.PUT_LINE('Total Employees: ' || emp_count);
END;
```

Lab 2:

```
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:

```
DECLARE
  dept_name VARCHAR2(50);
BEGIN
  SELECT department_name INTO dept_name FROM departments WHERE department_id = 10;
  IF dept_name = 'Sales' THEN
    DBMS_OUTPUT.PUT_LINE('Employee belongs to Sales department.');
  END IF;
END;
```

Lab 2:

```
DECLARE
  emp_name employees.name%TYPE;
BEGIN
  FOR emp IN (SELECT name FROM employees) LOOP
    DBMS_OUTPUT.PUT_LINE(emp.name);
  END LOOP;
END;
```

---

18. SQL Cursors

Lab 1:

```
DECLARE
  CURSOR emp_cursor IS SELECT * FROM employees;
  emp_row employees%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_row;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_row.name);
```

```
  END LOOP;
  CLOSE emp_cursor;
END;
```

Lab 2:

```
DECLARE
  CURSOR course_cursor IS SELECT * FROM courses;
  course_rec courses%ROWTYPE;
BEGIN
  OPEN course_cursor;
  LOOP
    FETCH course_cursor INTO course_rec;
    EXIT WHEN course_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(course_rec.course_name);
  END LOOP;
  CLOSE course_cursor;
END;
```

---

19. Rollback and Commit Savepoint

Lab 1:

```
START TRANSACTION;
SAVEPOINT sp1;
INSERT INTO members VALUES (6, 'John Doe', '2025-01-01', 'john@example.com');
ROLLBACK TO sp1;
```

Lab 2:

```
START TRANSACTION;
SAVEPOINT sp1;
INSERT INTO members VALUES (7, 'Jane Doe', '2025-02-01', 'jane@example.com');
COMMIT;
UPDATE members SET member_name = 'Jane D.' WHERE member_id = 7;
ROLLBACK;
```

1. Introduction to SQL

Q: Create a database called library_db and a table books with columns: book_id, title, author, publisher, year_of_publication, and price. Insert five records into the table.
A:

```
CREATE DATABASE library_db;

USE library_db;

CREATE TABLE books (
  book_id INT PRIMARY KEY,
  title VARCHAR(100),
  author VARCHAR(100),
  publisher VARCHAR(100),
```

```
  year_of_publication INT,
  price DECIMAL(10, 2)
);

INSERT INTO books VALUES
(1, 'Book A', 'Author X', 'Publisher A', 2010, 45.00),
(2, 'Book B', 'Author Y', 'Publisher B', 2015, 60.00),
(3, 'Book C', 'Author Z', 'Publisher C', 2020, 30.00),
(4, 'Book D', 'Author X', 'Publisher D', 2005, 25.00),
(5, 'Book E', 'Author Y', 'Publisher E', 2018, 50.00);
```

Q: Create a table members in library_db with columns: member_id, member_name, date_of_membership, and email. Insert five records into this table.
A:

```
CREATE TABLE members (
  member_id INT PRIMARY KEY,
  member_name VARCHAR(100),
  date_of_membership DATE,
  email VARCHAR(50)
);

INSERT INTO members VALUES
(1, 'Alice', '2019-03-12', 'alice@example.com'),
(2, 'Bob', '2021-07-01', 'bob@example.com'),
(3, 'Carol', '2018-05-20', 'carol@example.com'),
(4, 'Dave', '2023-01-11', 'dave@example.com'),
(5, 'Eve', '2020-11-25', 'eve@example.com');
```

---

2. SQL Syntax

Q: Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.
A:

```
SELECT * FROM members
WHERE date_of_membership < '2022-01-01'
ORDER BY date_of_membership;
```

Q: Write SQL queries to display the titles of books published by a specific author. Sort the results by year_of_publication in descending order.
A:

```
SELECT title FROM books
WHERE author = 'Author X'
ORDER BY year_of_publication DESC;
```

---

3. SQL Constraints

Q: Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.
A:

```
ALTER TABLE books
ADD CONSTRAINT chk_price CHECK (price > 0);
```

Q: Modify the members table to add a UNIQUE constraint on the email column.
A:

```
ALTER TABLE members
ADD CONSTRAINT unique_email UNIQUE (email);
```

---

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

Q: Create a table authors with columns: author_id, first_name, last_name, and country. Set author_id as the primary key.
A:

```
CREATE TABLE authors (
  author_id INT PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  country VARCHAR(50)
);
```

Q: Create a table publishers with columns: publisher_id, publisher_name, contact_number, and address. Set publisher_id as primary key and contact_number as unique.
A:

```
CREATE TABLE publishers (
  publisher_id INT PRIMARY KEY,
  publisher_name VARCHAR(100),
  contact_number VARCHAR(20) UNIQUE,
  address VARCHAR(255)
);
```

---

5. ALTER Command

Q: Add a new column genre to the books table. Update the genre for all existing records.
A:

```
ALTER TABLE books ADD genre VARCHAR(50);

UPDATE books SET genre = 'Fiction' WHERE book_id IN (1,2,3);
UPDATE books SET genre = 'Non-Fiction' WHERE book_id IN (4,5);
```

Q: Modify the members table to increase the length of the email column to 100 characters.
A:

```
ALTER TABLE members MODIFY email VARCHAR(100);
```

---

6. DROP Command

Q: Drop the publishers table from the database after verifying its structure.
A:

```
DESC publishers;
DROP TABLE publishers;
```

Q: Create a backup of the members table and then drop the original members table.
A:

```
CREATE TABLE members_backup AS SELECT * FROM members;
DROP TABLE members;
```

7. Data Manipulation Language (DML)

Lab 4:
Q: Insert three new authors into the authors table, then update the last name of one of the authors.
A:

```
INSERT INTO authors (author_id, first_name, last_name) VALUES (101, 'John', 'Smith');
INSERT INTO authors (author_id, first_name, last_name) VALUES (102, 'Emily', 'Johnson');
INSERT INTO authors (author_id, first_name, last_name) VALUES (103, 'Michael', 'Brown');

UPDATE authors SET last_name = 'Williams' WHERE author_id = 103;
```

Lab 5:
Q: Delete a book from the books table where the price is higher than $100.
A:

```
DELETE FROM books WHERE price > 100;
```

---

8. UPDATE Command

Lab 3:
Q: Update the year_of_publication of a book with a specific book_id.
A:

```
UPDATE books SET year_of_publication = 2022 WHERE book_id = 5;
```

Lab 4:
Q: Increase the price of all books published before 2015 by 10%.
A:

```
UPDATE books SET price = price * 1.10 WHERE year_of_publication < 2015;
```

---

9. DELETE Command

Lab 3:
Q: Remove all members who joined before 2020 from the members table.
A:

```
DELETE FROM members WHERE join_date < '2020-01-01';
```

Lab 4:
Q: Delete all books that have a NULL value in the author column.
A:

```
DELETE FROM books WHERE author IS NULL;
```

---

10. Data Query Language (DQL)

Lab 4:
Q: Write a query to retrieve all books with price between $50 and $100.
A:

```
SELECT * FROM books WHERE price BETWEEN 50 AND 100;
```

Lab 5:
Q: Retrieve the list of books sorted by author in ascending order and limit the results to the top 3 entries.
A:

```
SELECT * FROM books ORDER BY author ASC LIMIT 3;
```

---

11. Data Control Language (DCL)

Lab 3:
Q: Grant SELECT permission to a user named librarian on the books table.
A:

```
GRANT SELECT ON books TO librarian;
```

Lab 4:
Q: Grant INSERT and UPDATE permissions to the user admin on the members table.
A:

```
GRANT INSERT, UPDATE ON members TO admin;
```

---

12. REVOKE Command

Lab 3:
Q: Revoke the INSERT privilege from the user librarian on the books table.
A:

```
REVOKE INSERT ON books FROM librarian;
```

Lab 4:
Q: Revoke all permissions from user admin on the members table.
A:

```
REVOKE ALL PRIVILEGES ON members FROM admin;
```

---

13. Transaction Control Language (TCL)

Lab 3:
Q: Use COMMIT after inserting multiple records into the books table, then make another insertion and perform a ROLLBACK.
A:

```
BEGIN;

INSERT INTO books (book_id, title, author, price) VALUES (201, 'SQL Basics', 'John Smith', 45);
INSERT INTO books (book_id, title, author, price) VALUES (202, 'Advanced SQL', 'Emily Johnson', 75);

COMMIT;

INSERT INTO books (book_id, title, author, price) VALUES (203, 'SQL Mastery', 'Michael Brown', 95);

ROLLBACK;
```

Lab 4:
Q: Set a SAVEPOINT before making updates to the members table, perform some updates, and then roll back to the SAVEPOINT.
A:

```
BEGIN;

SAVEPOINT before_update;

UPDATE members SET status = 'inactive' WHERE last_login < '2022-01-01';
UPDATE members SET membership_type = 'basic' WHERE membership_type = 'premium';
```

```
ROLLBACK TO SAVEPOINT before_update;

COMMIT;
```

---

14. SQL Joins

Lab 3:
Q: Perform an INNER JOIN between books and authors tables to display the title of books and their respective authors' names.
A:

```
SELECT books.title, authors.first_name, authors.last_name
FROM books
INNER JOIN authors ON books.author_id = authors.author_id;
```

Lab 4:
Q: Use a FULL OUTER JOIN to retrieve all records from the books and authors tables, including those with no matching entries in the other table.
A:

```
SELECT books.title, authors.first_name, authors.last_name
FROM books
FULL OUTER JOIN authors ON books.author_id = authors.author_id;
```

---

15. SQL Group By

Lab 3:
Q: Group books by genre and display the total number of books in each genre.
A:

```
SELECT genre, COUNT(*) AS total_books
FROM books
GROUP BY genre;
```

Lab 4:
Q: Group members by the year they joined and find the number of members who joined each year.
A:

```
SELECT EXTRACT(YEAR FROM join_date) AS join_year, COUNT(*) AS total_members
FROM members
GROUP BY EXTRACT(YEAR FROM join_date);
```

16. SQL Stored Procedure

Lab 3:
Q: Write a stored procedure to retrieve all books by a particular author.
A:

```
CREATE PROCEDURE GetBooksByAuthor(IN authorName VARCHAR(100))
BEGIN
   SELECT * FROM books WHERE author = authorName;
END;
```

Lab 4:
Q: Write a stored procedure that takes book_id as an argument and returns the price of the book.
A:

```
CREATE PROCEDURE GetBookPrice(IN b_id INT)
BEGIN
   SELECT price FROM books WHERE book_id = b_id;
END;
```

---

17. SQL View

Lab 3:
Q: Create a view to show only the title, author, and price of books from the books table.
A:

```
CREATE VIEW book_summary AS
SELECT title, author, price FROM books;
```

Lab 4:
Q: Create a view to display members who joined before 2020.
A:

```
CREATE VIEW early_members AS
SELECT * FROM members WHERE join_date < '2020-01-01';
```

---

18. SQL Trigger

Lab 3:
Q: Create a trigger to automatically update the last_modified timestamp of the books table whenever a record is updated.
A:

```
CREATE TRIGGER update_last_modified
BEFORE UPDATE ON books
FOR EACH ROW
SET NEW.last_modified = NOW();
```

Lab 4:
Q: Create a trigger that inserts a log entry into a log_changes table whenever a DELETE operation is performed on the books table.
A:

```
CREATE TRIGGER log_book_deletion
AFTER DELETE ON books
```

```
FOR EACH ROW
INSERT INTO log_changes (action_type, book_id, action_time)
VALUES ('DELETE', OLD.book_id, NOW());
```

---

19. Introduction to PL/SQL

Lab 3:
Q: Write a PL/SQL block to insert a new book into the books table and display a confirmation message.
A:

```
BEGIN
   INSERT INTO books (book_id, title, author, price)
   VALUES (301, 'PLSQL Guide', 'Anna Scott', 59.99);
   DBMS_OUTPUT.PUT_LINE('Book inserted successfully.');
END;
```

Lab 4:
Q: Write a PL/SQL block to display the total number of books in the books table.
A:

```
DECLARE
   total_books NUMBER;
BEGIN
   SELECT COUNT(*) INTO total_books FROM books;
   DBMS_OUTPUT.PUT_LINE('Total number of books: ' || total_books);
END;
```

---

20. PL/SQL Syntax

Lab 3:
Q: Write a PL/SQL block to declare variables for book_id and price, assign values, and display the results.
A:

```
DECLARE
   book_id NUMBER := 101;
   price NUMBER := 49.99;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Book ID: ' || book_id || ', Price: $' || price);
END;
```

Lab 4:
Q: Write a PL/SQL block using constants and perform arithmetic operations on book prices.
A:

```
DECLARE
   CONSTANT discount_rate NUMBER := 0.10;
   original_price NUMBER := 100;
   final_price NUMBER;
BEGIN
```

```
    final_price := original_price - (original_price * discount_rate);
    DBMS_OUTPUT.PUT_LINE('Discounted price: $' || final_price);
END;
```

---

21. PL/SQL Control Structures

Lab 3:
Q: Write a PL/SQL block using IF-THEN-ELSE to check if a book's price is above $100 and print a message accordingly.
A:

```
DECLARE
    price NUMBER := 120;
BEGIN
    IF price > 100 THEN
        DBMS_OUTPUT.PUT_LINE('The book is expensive.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('The book is affordable.');
    END IF;
END;
```

Lab 4:
Q: Use a FOR LOOP in PL/SQL to display the details of all books one by one.
A:

```
DECLARE
    CURSOR book_cursor IS SELECT title, author, price FROM books;
    v_title books.title%TYPE;
    v_author books.author%TYPE;
    v_price books.price%TYPE;
BEGIN
    FOR book_record IN book_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('Title: ' || book_record.title ||
                             ', Author: ' || book_record.author ||
                             ', Price: $' || book_record.price);
    END LOOP;
END;
```

---

22. SQL Cursors

Lab 3:
Q: Write a PL/SQL block using an explicit cursor to fetch and display all records from the members table.
A:

```
DECLARE
    CURSOR member_cursor IS SELECT * FROM members;
    v_member members%ROWTYPE;
BEGIN
    OPEN member_cursor;
    LOOP
        FETCH member_cursor INTO v_member;
        EXIT WHEN member_cursor%NOTFOUND;
```

```
        DBMS_OUTPUT.PUT_LINE('Member ID: ' || v_member.member_id ||
                             ', Name: ' || v_member.name);
    END LOOP;
    CLOSE member_cursor;
END;
```

Lab 4:
Q: Create a cursor to retrieve books by a particular author and display their titles.
A:

```
DECLARE
    CURSOR author_books IS SELECT title FROM books WHERE author = 'John Smith';
    v_title books.title%TYPE;
BEGIN
    OPEN author_books;
    LOOP
        FETCH author_books INTO v_title;
        EXIT WHEN author_books%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Title: ' || v_title);
    END LOOP;
    CLOSE author_books;
END;
```

---

23. Rollback and Commit Savepoint

Lab 3:
Q: Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.
A:

```
START TRANSACTION;

INSERT INTO members (member_id, name, join_date) VALUES (401, 'David Green', '2025-07-01');

SAVEPOINT before_update;

UPDATE members SET name = 'David G.' WHERE member_id = 401;

ROLLBACK TO before_update;

COMMIT;
```

Lab 4:
Q: Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.
A:

```
START TRANSACTION;

INSERT INTO books (book_id, title, author, price) VALUES (501, 'Database Systems', 'Alan Turing',
60);
INSERT INTO books (book_id, title, author, price) VALUES (502, 'AI and SQL', 'Ada Lovelace', 85);

COMMIT;
```

```
START TRANSACTION;

SAVEPOINT update_point;

UPDATE books SET price = price + 10 WHERE book_id = 501;

ROLLBACK TO update_point;
```