Lab Exercise

A-1:

1. CREATE DATABASE School_db;

2. CREATE TABLE Students(student_id int PRIMARY KEY AUTO_INCREMENT, student_name varchar(50), age int, class int, address varchar(200));

3. INSERT INTO students (student_id,student_name,age,class,address) VALUES (1,'Vaishvi',21,7,'Rajkot');

4. SELECT * FROM students

A-2:

1. SELECT student_name,age FROM students;

2. SELECT * FROM students WHERE age > 10;

A-3:

1. CREATE TABLE teachers (teacher_id int PRIMARY KEY AUTO_INCREMENT, teacher_name varchar(50) NOT NULL, subject varchar(50), email varchar(100) UNIQUE);

2. ALTER TABLE students
   ADD CONSTRAINT teacher_id
   FOREIGN KEY (teacher_id)
   REFERENCES teachers(teacher_id);

A-4:

1. CREATE TABLE courses (course_id int PRIMARY KEY AUTO_INCREMENT, course_name varchar(50), course_credit int);

2. CREATE DATABASE university_db;

A-5:

1. ALTER TABLE courses ADD course_duration int;

2. ALTER TABLE courses DROP COLUMN course_credit;

A-6:

1. DROP TABLE students;

2. DROP TABLE teachers;

A-7:

1. INSERT INTO courses (course_id, course_name, course_duration) VALUES (1, 'Flutter', 3);

2. UPDATE courses SET course_duration = 8 WHERE course_id = 2;

3. DELETE FROM courses WHERE course_id = 1;

A-8:

1. SELECT * FROM courses;

2. SELECT * FROM courses ORDER BY course_duration DESC;

3. SELECT * FROM courses LIMIT 2;

A-9:

1. GRANT SELECT
   ON courses
   TO user1;

2. REVOKE SELECT
   ON courses
   TO user1;

A-10:

1. START TRANSACTION;

INSERT INTO courses (course_id, course_name, credits) VALUES (101, 'Mathematics', 3);
INSERT INTO courses (course_id, course_name, credits) VALUES (102, 'Physics', 4);

COMMIT;

2. START TRANSACTION;

   INSERT INTO courses (course_id, course_name, credits) VALUES (103, 'Chemistry', 3);
   INSERT INTO courses (course_id, course_name, credits) VALUES (104, 'Biology', 4);

   ROLLBACK;

3. START TRANSACTION;

   INSERT INTO courses (course_id, course_name, credits) VALUES (105, 'English', 2);

   SAVEPOINT before_update;
   UPDATE courses SET credits = 5 WHERE course_id = 101;
   ROLLBACK TO before_update;

   COMMIT;

A-11:

1. CREATE TABLE department (department_id INT PRIMARY KEY, department_name VARCHAR(100));

2. ALTER TABLE department
   ADD employee_id INT;

3. ALTER TABLE department
   ADD CONSTRAINT employee_id
   FOREIGN KEY (employee_id)
   REFERENCES employee(employee_id);

4. SELECT
    d.department_id,
    d.department_name,
    e.employee_id,
    e.employee_name
  FROM
    department d
  LEFT JOIN
    employee e ON d.employee_id = e.employee_id;

5. SELECT
    d.department_id,
    d.department_name,
    e.employee_id,
    e.employee_name
  FROM
    department d
  LEFT JOIN
    employee e ON d.employee_id = e.employee_id;

A-12:

1. SELECT *,
    COUNT(*) AS employee_count
  FROM
    employee
  GROUP BY
    Employee_name;

2. ALTER TABLE department ADD COLUMN salary int;

3. SELECT AVG (salary)
   FROM department;

A-13:

1. DELIMITER $$

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

   DELIMITER ;

2. DELIMITER $$

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

   DELIMITER ;

A-14:

1. DELIMITER $$

   CREATE PROCEDURE CreateEmployeeDepartmentView()
   BEGIN
      CREATE OR REPLACE VIEW EmployeeDepartmentView AS
      SELECT e.employee_id,
            e.name AS employee_name,
            e.salary,
            d.department_name
      FROM employees e
      JOIN departments d ON e.department_id = d.department_id;
   END$$

   DELIMITER ;

2. DELIMITER $$

   CREATE PROCEDURE UpdateEmployeeDepartmentView()
   BEGIN
      CREATE OR REPLACE VIEW EmployeeDepartmentView AS
      SELECT e.employee_id,
            e.name AS employee_name,
            e.salary,
            d.department_name
      FROM employees e
      JOIN departments d ON e.department_id = d.department_id
      WHERE e.salary >= 50000;
   END$$

   DELIMITER ;

A-15:

1. DELIMITER $$

   CREATE PROCEDURE CreateInsertLogTrigger()
   BEGIN
      CREATE TRIGGER LogNewEmployee
      AFTER INSERT ON employees
      FOR EACH ROW
      BEGIN
         INSERT INTO employee_log (employee_id, action)
         VALUES (NEW.employee_id, 'INSERT');
      END;
   END$$

   DELIMITER ;

2. DELIMITER $$

   CREATE PROCEDURE CreateUpdateTimestampTrigger()
   BEGIN
      CREATE TRIGGER UpdateEmployeeTimestamp
      BEFORE UPDATE ON employees
      FOR EACH ROW
      BEGIN
         SET NEW.last_modified = CURRENT_TIMESTAMP;
      END;
   END$$

   DELIMITER ;

A-16:

1.
```
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;
/
```

2.
```
DECLARE
    v_total_sales NUMBER;
BEGIN
    SELECT SUM(order_amount) INTO v_total_sales
    FROM orders;

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

A-17:

1.
```
DECLARE
   v_employee_id   employees.employee_id%TYPE := 101;  -- Change as needed
   v_department    employees.department%TYPE;
BEGIN
   SELECT department INTO v_department
   FROM employees
   WHERE employee_id = v_employee_id;

   IF v_department = 'HR' THEN
      DBMS_OUTPUT.PUT_LINE('Employee belongs to the HR department.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Employee does not belong to the HR department.');
   END IF;
END;
/
```

2.
```
DECLARE
   CURSOR emp_cursor IS
      SELECT name FROM employees;
BEGIN
   FOR emp_rec IN emp_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_rec.name);
   END LOOP;
END;
/
```

A-18:

1.
```
DECLARE
   CURSOR emp_cursor IS
      SELECT employee_id, name, department, salary
      FROM employees;

   v_emp_id     employees.employee_id%TYPE;
   v_name       employees.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_name, v_dept,
v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('ID: ' || v_emp_id || ', Name: ' ||
v_name ||
                           ', Department: ' || v_dept || ', Salary: ' || v_salary);
   END LOOP;
   CLOSE emp_cursor;
END;
/
```

2.
```
DECLARE
   CURSOR course_cursor IS
      SELECT course_id, course_name, duration
      FROM courses;

   v_course_id     courses.course_id%TYPE;
   v_course_name   courses.course_name%TYPE;
   v_duration      courses.duration%TYPE;
```

```
BEGIN
   OPEN course_cursor;
   LOOP
      FETCH course_cursor INTO v_course_id, v_course_name,
v_duration;
      EXIT WHEN course_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Course ID: ' || v_course_id || ',
Name: ' || v_course_name ||
                           ', Duration: ' || v_duration || ' hours');
   END LOOP;
   CLOSE course_cursor;
END;
/
```

A-19:

1. BEGIN

```
-- Start Transaction
INSERT INTO employees (employee_id, name, department, salary)
VALUES (201, 'Alice Smith', 'Finance', 60000);

SAVEPOINT emp_insert_savepoint;

INSERT INTO employees (employee_id, name, department, salary)
VALUES (202, 'Bob Johnson', 'HR', 55000);

-- Something goes wrong; rollback only the second insert
ROLLBACK TO emp_insert_savepoint;

-- Commit the first insert
COMMIT;

DBMS_OUTPUT.PUT_LINE('Transaction rolled back to savepoint.
First insert committed.');
END;
```

/

2. BEGIN
-- First part of the transaction
INSERT INTO employees (employee_id, name, department, salary)
VALUES (203, 'Charlie Brown', 'IT', 70000);

SAVEPOINT part1_done;

-- Second part of the transaction
INSERT INTO employees (employee_id, name, department, salary)
VALUES (204, 'Diana Prince', 'Marketing', 52000);

-- Commit first insert (Charlie)
COMMIT;

-- Something goes wrong with second insert
ROLLBACK TO part1_done;

DBMS_OUTPUT.PUT_LINE('First insert committed. Second insert rolled back.');
END;
/

Extra Lab - 1:

A-1:

1. CREATE DATABASE library_db;

2. CREATE TABLE books (
    book_id INT PRIMARY KEY,
    title VARCHAR(200),

```
    author VARCHAR(100),
    publisher VARCHAR(100),
    year_of_publication INT,
    price DECIMAL(8, 2)
);
```

3. INSERT INTO books (book_id, title, author, publisher, year_of_publication, price) VALUES
   (1, 'The Great Gatsby', 'F. Scott', 'Scribner', 1925, 10.99),

A-2:

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

2. INSERT INTO members (member_id, member_name, date_of_membership, email) VALUES
   (1, 'Alice Johnson', '2021-01-15', 'alice.johnson@example.com'),

Extra Lab - 2:

A-1:
```
SELECT *
FROM members
WHERE date_of_membership < '2022-01-01'
ORDER BY date_of_membership;
```

A-2:
```
SELECT title
FROM books
WHERE author = 'George Orwell'   ORDER BY
year_of_publication DESC;
```

Extra Lab - 3:

A-1:

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

A-2:

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

Extra Lab - 4:

A-1:

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

A-2:

```
CREATE TABLE publishers (
    publisher_id INT PRIMARY KEY,
    publisher_name VARCHAR(100),
    contact_number VARCHAR(20) UNIQUE,
    address VARCHAR(150)
);
```

Extra Lab - 5:

A-1:

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

UPDATE books SET genre = 'Classic';
```

A-2:

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

ALTER TABLE members
ALTER COLUMN email TYPE VARCHAR(100);
```

Extra Lab - 6:

A-1:

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

A-2:

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

Extra Lab - 7:

A-1:

```
INSERT INTO authors (author_id, first_name, last_name) VALUES
(101, 'John', 'Smith');
```

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

A-2:

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

Extra Lab - 8:

A-1:

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

A-2:

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

Extra Lab - 9:

A-1:

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

A-2:

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

Extra Lab - 10:

A-1:

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

A-2:

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

Extra Lab - 11:

A-1:

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

A-2:

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

Extra Lab - 12:

A-1:

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

A-2:

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

Extra Lab - 13:

A-1:

```
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;
```

A-2:

```
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;
```

Extra Lab - 14:

A-1:

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

A-2:

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

Extra Lab - 15:

A-1:

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

A-2:

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

Extra Lab - 16:

A-1:

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

A-2:

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

Extra Lab - 17:

A-1:

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

A-2:

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

Extra Lab - 18:

A-1:

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

A-2:

```
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());
```

Extra Lab - 19:

A-1:

```
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;
```

A-2:

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

Extra Lab - 20:

A-1:

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

A-2:

```
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;
```

Extra Lab - 21:

A-1:

```
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;
```

A-1:

```
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;
```

Extra Lab - 22:

A-1:

```
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;
```

A-2:

```
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;
```

Extra Lab - 23:

A-1:

```
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;
```

A-2:

```
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;
```