MODULE-4 PRACTICAL

Introduction to dbms

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.

```
mysql> -- Step 1: Create the database
mysql> CREATE DATABASE school_db;
Query OK, 1 row affected (0.02 sec)

mysql>
mysql> -- Step 2: Use the newly created database
mysql> USE school_db;
Database changed
mysql>
mysql> -- Step 3: Create the 'students' table
mysql> CREATE TABLE students (
    ->     student_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     student_name VARCHAR(100) NOT NULL,
    ->     age INT,
    ->     class VARCHAR(50),
    ->     address VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
```

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

```
mysql> INSERT INTO students (student_name, age, class, address) VALUES
    -> ('Amit Sharma', 15, '10th Grade', '123 Green Street'),
    -> ('Neha Patel', 14, '9th Grade', '45 Blue Avenue'),
    -> ('Rohan Singh', 16, '11th Grade', '78 Red Road'),
    -> ('Priya Deshmukh', 15, '10th Grade', '56 Yellow Lane'),
    -> ('Vikram Rao', 17, '12th Grade', '89 Orange Street');
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
```

2.sql syntax

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

```
mysql> SELECT student_name, age
    -> FROM students;
+----------------+------+
| student_name   | age  |
+----------------+------+
| Amit Sharma    |   15 |
| Neha Patel     |   14 |
| Rohan Singh    |   16 |
| Priya Deshmukh |   15 |
| Vikram Rao     |   17 |
+----------------+------+
5 rows in set (0.01 sec)

mysql>
```

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

```
ysql> SELECT *
   -> FROM students
   -> WHERE age > 10;
+------------+----------------+------+------------+------------------+
| student_id | student_name   | age  | class      | address          |
+------------+----------------+------+------------+------------------+
|          1 | Amit Sharma    |   15 | 10th Grade | 123 Green Street |
|          2 | Neha Patel     |   14 | 9th Grade  | 45 Blue Avenue   |
|          3 | Rohan Singh    |   16 | 11th Grade | 78 Red Road      |
|          4 | Priya Deshmukh |   15 | 10th Grade | 56 Yellow Lane   |
|          5 | Vikram Rao     |   17 | 12th Grade | 89 Orange Street |
+------------+----------------+------+------------+------------------+
5 rows in set (0.00 sec)
```

3. SQL Constraints

1. Create a table teachers with the following columns: teacher_id (Primary Key),

```
mysql> CREATE TABLE teachers (
    ->     teacher_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     teacher_name VARCHAR(100) NOT NULL,
    ->     subject VARCHAR(100) NOT NULL,
    ->     email VARCHAR(100) UNIQUE
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
```

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

```
mysql> ALTER TABLE students
    -> ADD COLUMN teacher_id INT,
    -> ADD CONSTRAINT fk_teacher
    -> FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> DESC teachers;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| teacher_id   | int          | NO   | PRI | NULL    | auto_increment |
| teacher_name | varchar(100) | NO   |     | NULL    |                |
| subject      | varchar(100) | NO   |     | NULL    |                |
| email        | varchar(100) | YES  | UNI | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
4 rows in set (0.10 sec)

mysql>
```

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

1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

```
mysql> CREATE TABLE courses (
    ->     course_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     course_name VARCHAR(100) NOT NULL,
    ->     course_credits INT NOT NULL
    -> );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> DESC courses;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| course_id       | int          | NO   | PRI | NULL    | auto_increment |
| course_name     | varchar(100) | NO   |     | NULL    |                |
| course_credits  | int          | NO   |     | NULL    |                |
| course_duration | varchar(50)  | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql>
```

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

```
mysql> CREATE DATABASE university_db;
Query OK, 1 row affected (0.00 sec)
```

```
ysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| library_db         |
| mysql              |
| performance_schema |
| project_db         |
| school_db          |
| statement          |
| student            |
| student1           |
| sys                |
| university_db      |
| user_db            |
| user_db1           |
+--------------------+
13 rows in set (0.01 sec)
```

5. ALTER Command

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

```
mysql> ALTER TABLE courses
    -> ADD COLUMN course_duration VARCHAR(50);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> DESC courses;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| course_id      | int         | NO   | PRI | NULL    | auto_increment |
| course_name    | varchar(100)| NO   |     | NULL    |                |
| course_credits | int         | NO   |     | NULL    |                |
| course_duration| varchar(50) | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

(2) . How can you add, modify, and drop columns from a table using ALTER?

Drop columns:

```
ysql> ALTER table Courses
    -> DROP course_credits;
uery OK, 0 rows affected (0.04 sec)
ecords: 0  Duplicates: 0  Warnings: 0

ysql> ^C
ysql> DESC courses;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| course_id      | int         | NO   | PRI | NULL    | auto_increment |
| course_name    | varchar(100)| NO   |     | NULL    |                |
| course_duration| varchar(50) | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
 rows in set (0.00 sec)

ysql>
```

6.drop command

1. What is the function of the DROP command in SQL?

```
mysql> DROP TABLE courses;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+----------------------+
| Tables_in_school_db  |
+----------------------+
| students             |
| teachers             |
+----------------------+
2 rows in set (0.01 sec)

mysql>
```

2. What are the implications of dropping a table from a database?

```
mysql> DROP TABLE students;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+--------------------+
| Tables_in_school_db |
+--------------------+
| teachers           |
+--------------------+
1 row in set (0.00 sec)

mysql>
```

7.Data manipulation language

1. Define the INSERT, UPDATE, and DELETE commands in SQL

```
mysql> INSERT INTO courses (course_name, course_credits) VALUES
    -> ('Mathematics', 3),
    -> ('Physics', 4),
    -> ('Chemistry', 4),
    -> ('Computer Science', 5),
    -> ('English Literature', 2);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> DESC courses;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| course_id      | int          | NO   | PRI | NULL    | auto_increment |
| course_name    | varchar(100) | NO   |     | NULL    |                |
| course_credits | int          | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
3 rows in set (0.02 sec)

mysql> SELECT *from courses;
+-----------+--------------------+----------------+
| course_id | course_name        | course_credits |
+-----------+--------------------+----------------+
|         1 | Mathematics        |              3 |
|         2 | Physics            |              4 |
|         3 | Chemistry          |              4 |
|         4 | Computer Science   |              5 |
|         5 | English Literature |              2 |
+-----------+--------------------+----------------+
5 rows in set (0.00 sec)

mysql>
```

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

```
    -> WHERE course_id = 1' at line 2
mysql>  UPDATE courses
    ->  SET course_credits = 5
    ->   WHERE course_id = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT *from courses;
+-----------+--------------------+----------------+
| course_id | course_name        | course_credits |
+-----------+--------------------+----------------+
|         1 | Mathematics        |              5 |
|         2 | Physics            |              4 |
|         3 | Chemistry          |              4 |
|         4 | Computer Science   |              5 |
|         5 | English Literature |              2 |
+-----------+--------------------+----------------+
5 rows in set (0.00 sec)

mysql>
```

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

```
ysql> DELETE FROM courses
    -> WHERE course_id = 5;
uery OK, 1 row affected (0.07 sec)

ysql> SELECT *from courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
+-----------+------------------+----------------+
 rows in set (0.01 sec)

ysql>
```

8. Data Query Language (DQL)

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

```
ysql> SELECT *
    -> FROM courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
+-----------+------------------+----------------+
 rows in set (0.00 sec)

ysql>
```

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

```
mysql> SELECT * from courses
    -> ORDER BY course_credits DESC;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         4 | Computer Science |              5 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
+-----------+------------------+----------------+
4 rows in set (0.01 sec)

mysql>
```

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

```
mysql> SELECT * from courses
    -> ORDER BY course_credits DESC
    -> LIMIT 2;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         4 | Computer Science |              5 |
+-----------+------------------+----------------+
2 rows in set (0.00 sec)

mysql>
```

9. Data Control Language (DCL)

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

```
mysql> CREATE USER 'USER2' IDENTIFIED BY 'user2';
Query OK, 0 rows affected (0.01 sec)

mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| USER1            |
| USER2            |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
| user1            |
| user2            |
+------------------+
8 rows in set (0.00 sec)

mysql>
```

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

```
mysql> GRANT INSERT on school_db.courses to 'USER1';
Query OK, 0 rows affected (0.00 sec)

mysql> REVOKE INSERT on school_db.courses FROM  'USER1' ;
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT INSERT on school_db.courses to 'USER2';
Query OK, 0 rows affected (0.00 sec)
```

---

10. Transaction Control Language (TCL)

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

```
mysql> INSERT INTO COURSES VALUES(5,'se',4),
    -> ( 6,'java', 6);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
|         5 | se               |              4 |
|         6 | java             |              6 |
+-----------+------------------+----------------+
6 rows in set (0.00 sec)

mysql>
```

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

```
mysql> INSERT INTO courses values(7,'english',3),
    -> (8,'gujrati',5);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              5 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
|         5 | se               |              4 |
|         6 | java             |              6 |
|         7 | english          |              3 |
|         8 | gujrati          |              5 |
+-----------+------------------+----------------+
8 rows in set (0.00 sec)
```

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

```
mysql> UPDATE courses
    -> SET course_credits = 6
    -> WHERE course_name = 'Mathematics';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT *from courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              6 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
|         5 | se               |              4 |
|         6 | java             |              6 |
|         7 | english          |              3 |
|         8 | gujrati          |              5 |
+-----------+------------------+----------------+
8 rows in set (0.00 sec)

mysql>
```

11. SQL Joins

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

```
mysql> CREATE TABLE departments (
    ->     dept_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     dept_name VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO departments (dept_name) VALUES
    -> ('HR'),
    -> ('Finance'),
    -> ('IT'),
    -> ('Marketing');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select *from department;
ERROR 1146 (42S02): Table 'university_db.department' doesn't exist
mysql> select *from departments;
+---------+-----------+
| dept_id | dept_name |
+---------+-----------+
|       1 | HR        |
|       2 | Finance   |
|       3 | IT        |
|       4 | Marketing |
+---------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> INSERT INTO employees (emp_name, dept_id) VALUES
    -> ('Alice', 1),
    -> ('Bob', 3),
    -> ('Charlie', 2),
    -> ('David', 3),
    -> ('Eve', 4);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select *from employees;
+--------+----------+---------+
| emp_id | emp_name | dept_id |
+--------+----------+---------+
|      1 | Alice    |       1 |
|      2 | Bob      |       3 |
|      3 | Charlie  |       2 |
|      4 | David    |       3 |
|      5 | Eve      |       4 |
+--------+----------+---------+
5 rows in set (0.00 sec)

mysql>
```

RIGHT JOIN

```
mysql> SELECT e.emp_id, e.emp_name, d.dept_name
    -> FROM employees e
    -> INNER JOIN departments d ON e.dept_id = d.dept_id;
+--------+----------+-----------+
| emp_id | emp_name | dept_name |
+--------+----------+-----------+
|      1 | Alice    | HR        |
|      3 | Charlie  | Finance   |
|      2 | Bob      | IT        |
|      4 | David    | IT        |
|      5 | Eve      | Marketing |
+--------+----------+-----------+
5 rows in set (0.01 sec)

mysql>
```

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

```
mysql> SELECT d.dept_id, d.dept_name, e.emp_id, e.emp_name
    -> FROM departments d
    -> LEFT JOIN employees e ON d.dept_id = e.dept_id;
+---------+-----------+--------+----------+
| dept_id | dept_name | emp_id | emp_name |
+---------+-----------+--------+----------+
|       1 | HR        |      1 | Alice    |
|       2 | Finance   |      3 | Charlie  |
|       3 | IT        |      2 | Bob      |
|       3 | IT        |      4 | David    |
|       4 | Marketing |      5 | Eve      |
+---------+-----------+--------+----------+
5 rows in set (0.00 sec)

mysql>
```

12. SQL Group By

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

```
mysql> SELECT d.dept_name, COUNT(e.emp_id) AS employee_count
    -> FROM departments d
    -> LEFT JOIN employees e ON d.dept_id = e.dept_id
    -> GROUP BY d.dept_name;
+-----------+----------------+
| dept_name | employee_count |
+-----------+----------------+
| HR        |              1 |
| Finance   |              1 |
| IT        |              2 |
| Marketing |              1 |
+-----------+----------------+
4 rows in set (0.01 sec)
```

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

```
mysql> SELECT d.dept_name, COUNT(e.emp_id) AS employee_count
    -> FROM departments d
    -> LEFT JOIN employees e ON d.dept_id = e.dept_id
    -> GROUP BY d.dept_name;
+-----------+----------------+
| dept_name | employee_count |
+-----------+----------------+
| HR        |              1 |
| Finance   |              1 |
| IT        |              2 |
| Marketing |              1 |
+-----------+----------------+
4 rows in set (0.00 sec)
```

13. SQL Stored Procedure

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

```
mysql> select *from employees;
+--------+----------+---------+
| emp_id | emp_name | dept_id |
+--------+----------+---------+
|      1 | Alice    |       1 |
|      2 | Bob      |       3 |
|      3 | Charlie  |       2 |
|      4 | David    |       3 |
|      5 | Eve      |       4 |
+--------+----------+---------+
5 rows in set (0.00 sec)

mysql>
```

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

```
ysql> CREATE PROCEDURE GetCourseDetails(IN courseID INT)
    -> BEGIN
    ->     SELECT course_id, name, credits, course_duration
    ->     FROM courses
    ->     WHERE course_id = courseID;
    -> END $$
Query OK, 0 rows affected (0.01 sec)

ysql>
ysql> DELIMITER ;
ysql> select *from courses;
+-----------+------------------+----------------+
| course_id | course_name      | course_credits |
+-----------+------------------+----------------+
|         1 | Mathematics      |              6 |
|         2 | Physics          |              4 |
|         3 | Chemistry        |              4 |
|         4 | Computer Science |              5 |
|         5 | se               |              4 |
|         6 | java             |              6 |
|         7 | english          |              3 |
|         8 | gujrati          |              5 |
+-----------+------------------+----------------+
8 rows in set (0.00 sec)
```

14. SQL View

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

```
mysql> CREATE VIEW EmployeeDeptView AS
    -> SELECT e.emp_id, e.emp_name, d.dept_name
    -> FROM employees e
    -> INNER JOIN departments d ON e.dept_id = d.dept_id;
Query OK, 0 rows affected (0.01 sec)

mysql> select *from employees;
+--------+----------+---------+
| emp_id | emp_name | dept_id |
+--------+----------+---------+
|      1 | Alice    |       1 |
|      2 | Bob      |       3 |
|      3 | Charlie  |       2 |
|      4 | David    |       3 |
|      5 | Eve      |       4 |
+--------+----------+---------+
5 rows in set (0.00 sec)
```

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

```
mysql> CREATE OR REPLACE VIEW EmployeeDeptView 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;
Query OK, 0 rows affected (0.01 sec)

mysql> select *from employees;
+--------+----------+---------+----------+
| emp_id | emp_name | dept_id | salary   |
+--------+----------+---------+----------+
|      1 | Alice    |       1 | 50000.00 |
|      2 | Bob      |       3 | 60000.00 |
|      3 | Charlie  |       2 | 55000.00 |
|      4 | David    |       3 | 62000.00 |
|      5 | Eve      |       4 | 58000.00 |
+--------+----------+---------+----------+
5 rows in set (0.00 sec)
```

15. SQL Triggers

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

```
mysql> CREATE TABLE employee_log (
    ->     log_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     emp_id INT,
    ->     emp_name VARCHAR(50),
    ->     dept_id INT,
    ->     action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ->     action_type VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER after_employee_insert
    -> AFTER INSERT ON employees
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO employee_log (emp_id, emp_name, dept_id, action_type)
    ->     VALUES (NEW.emp_id, NEW.emp_name, NEW.dept_id, 'INSERT');
    -> END $$
Query OK, 0 rows affected (0.01 sec)
```

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

```
mysql> ALTER TABLE employees
    -> ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER before_employee_update
    -> BEFORE UPDATE ON employees
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.last_modified = CURRENT_TIMESTAMP;
    -> END $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

16. Introduction to PL/SQL

1: Write a PL/SQL block to print the total number of employees from the employees table.

```
mysql> SELECT COUNT(*) AS total_employees
    -> FROM employees;
+-----------------+
| total_employees |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.01 sec)

mysql>
```

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

```
mysql>
mysql> -- Calculate total sales
mysql> SELECT SUM(order_amount) AS total_sales
    -> FROM orders;
+-------------+
| total_sales |
+-------------+
|      901.25 |
+-------------+
1 row in set (0.00 sec)

mysql>
```

17. PL/SQL Control Structures

1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

```
   SELECT dept_id INTO deptID
   FROM employees
   WHERE emp_id = empID;
27 (42000): Undeclared variable: deptID
ELIMITER $$

REATE PROCEDURE CheckEmployeeDept(IN empID INT)
EGIN
   DECLARE deptID INT;

   SELECT dept_id INTO deptID
   FROM employees
   WHERE emp_id = empID;

   IF deptID = 1 THEN
       SELECT 'Employee belongs to HR department.' AS message;
   ELSEIF deptID = 2 THEN
       SELECT 'Employee belongs to Finance department.' AS message;
   ELSEIF deptID = 3 THEN
       SELECT 'Employee belongs to IT department.' AS message;
   ELSE
       SELECT 'Employee belongs to another department.' AS message;
   END IF;
ND $$
, 0 rows affected (0.01 sec)


ELIMITER ;
```

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

```
CREATE PROCEDURE ListEmployeeNames()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE emp_name_var VARCHAR(50);
    DECLARE emp_cursor CURSOR FOR SELECT emp_name FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN emp_cursor;

    loop_label: LOOP
        FETCH emp_cursor INTO emp_name_var;
        IF done = 1 THEN
            LEAVE loop_label;
        END IF;
        SELECT emp_name_var AS employee_name;
    END LOOP;

    CLOSE emp_cursor;
END $$
OK, 0 rows affected (0.01 sec)


DELIMITER ;
```

18. SQL Cursors

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

```
DELIMITER $$

CREATE PROCEDURE ListEmployeeDetails()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_emp_id INT;
    DECLARE v_emp_name VARCHAR(50);
    DECLARE v_dept_id INT;

    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, dept_id FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO v_emp_id, v_emp_name, v_dept_id;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;
        SELECT v_emp_id AS emp_id, v_emp_name AS emp_name, v_dept_id AS dept_id;
    END LOOP;

    CLOSE emp_cursor;
END $$
OK, 0 rows affected (0.01 sec)
```

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

```
CREATE PROCEDURE ListCourses()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_course_id INT;
    DECLARE v_name VARCHAR(100);
    DECLARE v_credits INT;
    DECLARE v_duration INT;

    DECLARE course_cursor CURSOR FOR
        SELECT course_id, name, credits, course_duration FROM courses;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN course_cursor;

    read_loop: LOOP
        FETCH course_cursor INTO v_course_id, v_name, v_credits, v_duration;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;
        SELECT v_course_id AS course_id, v_name AS name, v_credits AS credits, v_duration AS course_duration;
    END LOOP;

    CLOSE course_cursor;
END $$
OK, 0 rows affected (0.01 sec)
```

19. Rollback and Commit Savepoint

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

```
mysql> SAVEPOINT before_insert;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO courses (course_id, course_name, course_credits) VALUES
    -> (9, 'History', 3),
    -> (10, 'Geography', 2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> ROLLBACK TO SAVEPOINT before_insert;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from courses;
+-----------+-----------------+----------------+
| course_id | course_name     | course_credits |
+-----------+-----------------+----------------+
|         1 | Mathematics     |              6 |
|         2 | Physics         |              4 |
|         3 | Chemistry       |              4 |
|         4 | Computer Science |              5 |
|         5 | se              |              4 |
|         6 | java            |              6 |
|         7 | english         |              3 |
|         8 | gujrati         |              5 |
+-----------+-----------------+----------------+
8 rows in set (0.00 sec)
```

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

```
mysql>
mysql> ROLLBACK TO SAVEPOINT sp_before_insert;
ERROR 1305 (42000): SAVEPOINT sp_before_insert does not exist
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SAVEPOINT sp_before_insert;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO courses (course_name, course_credits) VALUES
    -> ('History', 3);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> INSERT INTO courses (course_name, course_credits) VALUES
    -> ('Geography', 2),
    -> ('Civics', 2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> ROLLBACK TO SAVEPOINT sp_before_insert;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```