S1)

1.

1. SQL command to show the Regd. No of Pradeep:

```sql
SELECT REGDINO
FROM STUDENT
WHERE NAME = 'Pradeep';
```

2.

2. SQL command to display the Name and Branch of Regd. No 0002:

```sql
SELECT NAME, BRANCH
FROM STUDENT
WHERE REGDINO = '0002';
```

3.

3. SQL command to count the number of rows existing in the STUDENT table:

```sql
SELECT COUNT(*) AS row_count
FROM STUDENT;
```

4.

4. SQL command to add another column address in the STUDENT table:

```sql
ALTER TABLE STUDENT
ADD COLUMN ADDRESS VARCHAR(50);
```

5.

5. SQL command to change the branch of Ram from CSE to ETC:

```sql
UPDATE STUDENT
SET BRANCH = 'ETC'
WHERE NAME = 'Ram';
```

S2)

1.

1. The SQL command to delete the record of Ram from the STUDENT table would be:

```sql
DELETE FROM STUDENT WHERE Name = 'Ram';
```

2.

2. The SQL command to show all records whose name starts with "R" from the STUDENT table would be:

```sql
SELECT * FROM STUDENT WHERE Name LIKE 'R%';
```

3.

3. The SQL command to update the name of "Pradeep" to "Pradeep Kumar" in the STUDENT table would be:

```sql
UPDATE STUDENT SET Name = 'Pradeep Kumar' WHERE Name = 'Pradeep';
```

4.

4. To add another column called "Address" to the STUDENT table, you would need to use the ALTER TABLE statement to modify the table structure:

```sql
ALTER TABLE STUDENT ADD Address VARCHAR(255);
```

To update all addresses in the STUDENT table, you can use the UPDATE statement:

```sql
UPDATE STUDENT SET Address = 'New Address' WHERE 1=1;
```

5.

5. To find the total registration count of each branch in the STUDENT table, you can use the following SQL query:

```sql
SELECT Branch, COUNT(*) AS RegistrationCount
FROM STUDENT
GROUP BY Branch;
```

This query will group the records by branch and calculate the count of registrations for each branch, displaying the branch name and its corresponding registration count.

S3)

1.

1. Find the name of all salespeople that have an order with Samsonic.

```sql
SELECT DISTINCT sp.Name
FROM Salesperson sp
JOIN Product p ON sp.SID = p.SID
WHERE p.Name = 'Samsonic';
```

2.

2. The names of all salespeople that do not have any order with Samsonic.

```sql
SELECT DISTINCT sp.Name
FROM Salesperson sp
LEFT JOIN Product p ON sp.SID = p.SID
WHERE p.PID IS NULL OR p.Name <> 'Samsonic';
```

3.

3. The names of salespeople that have 2 or more orders.

```sql
SELECT sp.Name
FROM Salesperson sp
JOIN Product p ON sp.SID = p.SID
GROUP BY sp.Name
HAVING COUNT(p.PID) >= 2;
```

4.

4. Display all salespeople with age less than 40.

```sql
SELECT *
FROM Salesperson
WHERE Age < 40;
```

5.

5. Find the ID and name of the salesperson who is selling the product Panasonic.

```sql
SELECT sp.SID, sp.Name
FROM Salesperson sp
JOIN Product p ON sp.SID = p.SID
WHERE p.Name = 'Panasonic';
```

S5)

1.

1. Find the distinct pnames of all parts:

```sql
SELECT DISTINCT pname FROM Parts;
```

2.

2. Alter the data types of sname as varchar(30):

```sql
ALTER TABLE Suppliers
ALTER COLUMN sname TYPE varchar(30);
```

3.

3. Find out the supplier who is supplying part "Keyboard" whose cost is 5000:

```sql
SELECT Suppliers.sname
FROM Suppliers
JOIN Catalog ON Suppliers.sid = Catalog.sid
JOIN Parts ON Catalog.pid = Parts.pid
WHERE Parts.pname = 'Keyboard' AND Catalog.cost = 5000;
```

4.

4. Remove all parts whose name is "Mouse":

```sql
DELETE FROM Parts
WHERE pname = 'Mouse';
```

5.

5. List all suppliers whose name starts with "S" in descending order:

sql Copy code

```sql
SELECT *
FROM Suppliers
WHERE sname LIKE 'S%'
ORDER BY sname DESC;
```

S6)

```
-- Creating the 'dept' table
CREATE TABLE dept (
  deptno INT PRIMARY KEY,
  dname VARCHAR(50),
  mgreno INT
);

-- Creating the 'emp' table
CREATE TABLE emp (
  eno INT PRIMARY KEY,
  ename VARCHAR(50),
  bdate DATE,
  title VARCHAR(50),
  salary DECIMAL(10,2),
  deptno INT,
  FOREIGN KEY (deptno) REFERENCES dept(deptno)
);

-- Creating the 'proj' table
CREATE TABLE proj (
```

```
  pno INT PRIMARY KEY,
  pname VARCHAR(50),
  budget DECIMAL(10,2),
  deptno INT,
  FOREIGN KEY (deptno) REFERENCES dept(deptno)
);

-- Creating the 'workson' table
CREATE TABLE workson (
  eno INT,
  pno INT,
  responsibility VARCHAR(50),
  hours INT,
  FOREIGN KEY (eno) REFERENCES emp(eno),
  FOREIGN KEY (pno) REFERENCES proj(pno)
);

-- Inserting values into 'dept' table
INSERT INTO dept (deptno, dname, mgreno) VALUES
(1, 'D1', 100),
(2, 'D2', 200),
(3, 'D3', 300);

-- Inserting values into 'emp' table
INSERT INTO emp (eno, ename, bdate, title, salary, deptno) VALUES
(1, 'John', '1990-05-15', 'EE', 40000, 1),
(2, 'Jane', '1995-10-20', 'SA', 45000, 1),
(3, 'Mike', '1992-08-08', 'EE', 55000, 2),
(4, 'Sarah', '1998-03-01', 'Manager', 60000, 2),
(5, 'David', '1993-06-25', 'SA', 38000, 3),
(6, 'Emily', '1997-12-12', 'Manager', 70000, 3);
```

-- Inserting values into 'proj' table

INSERT INTO proj (pno, pname, budget, deptno) VALUES

(1, 'Project A', 150000, 1),

(2, 'Project B', 90000, 2),

(3, 'Project C', 120000, 2),

(4, 'Project D', 80000, 3),

(5, 'Project E', 200000, 3);

-- Inserting values into 'workson' table

INSERT INTO workson (eno, pno, responsibility, hours) VALUES

(1, 1, 'Developer', 40),

(2, 1, 'Manager', 30),

(3, 2, 'Developer', 20),

(4, 3, 'Manager', 25),

(5, 4, 'Developer', 15),

(6, 5, 'Manager', 35);

1.

1. SQL query to return project number and name for projects with a budget greater than Rs.100,000:

sql Copy code

```sql
SELECT pno, pname
FROM proj
WHERE budget > 100000;
```

2.

2. SQL query to return all 'workson' records where hours worked is less than 10 and the responsibility is 'Manager':

```sql
SELECT *
FROM workson
WHERE hours < 10 AND responsibility = 'Manager';
```

3.

3. SQL query to return employees (number and name only) who have a title of 'EE' or 'SA' and make more than Rs.35,000:

```sql
SELECT eno, ename
FROM emp
WHERE title IN ('EE', 'SA') AND salary > 35000;
```

4.

4. SQL query to return employees' names only in department 'D1' ordered by decreasing salary:

```sql
SELECT ename
FROM emp
WHERE deptno = 1
ORDER BY salary DESC;
```

5.

5. SQL query to list manager numbers whose department name is 'Production':

sql Copy code

```sql
SELECT e.eno AS manager_number
FROM emp e
JOIN dept d ON e.deptno = d.deptno
WHERE d.dname = 'Production' AND e.title = 'Manager';
```

S7}

```
CREATE TABLE dept (
  dno INT PRIMARY KEY,
  dname VARCHAR(50),
  mgreno INT
);

CREATE TABLE emp (
  eno INT PRIMARY KEY,
  ename VARCHAR(50),
  bdate DATE,
  title VARCHAR(50),
  salary DECIMAL(10, 2),
  dno INT,
  FOREIGN KEY (dno) REFERENCES dept(dno)
);

CREATE TABLE proj (
  pno INT PRIMARY KEY,
  pname VARCHAR(50),
```

```
  budget DECIMAL(10, 2),
  dno INT,
  FOREIGN KEY (dno) REFERENCES dept(dno)
);

CREATE TABLE workson (
  eno INT,
  pno INT,
  responsibility VARCHAR(50),
  hours INT,
  FOREIGN KEY (eno) REFERENCES emp(eno),
  FOREIGN KEY (pno) REFERENCES proj(pno)
);

INSERT INTO dept (dno, dname, mgreno) VALUES
  (1, 'Department A', 1001),
  (2, 'Department B', 1002),
  (3, 'Department C', 1003);

INSERT INTO emp (eno, ename, bdate, title, salary, dno) VALUES
  (10001, 'John Doe', '1990-01-01', 'Manager', 5000, 1),
  (10002, 'Jane Smith', '1995-05-10', 'Developer', 3000, 1),
  (10003, 'Michael Johnson', '1992-08-15', 'Analyst', 4000, 2),
  (10004, 'Emily Davis', '1997-03-20', 'Designer', 3500, 2),
  (10005, 'Robert Brown', '1988-12-05', 'Tester', 3200, 3);

INSERT INTO proj (pno, pname, budget, dno) VALUES
  (1, 'Project X', 80000, 1),
  (2, 'Project Y', 60000, 2),
  (3, 'Project Z', 40000, 3);
```

INSERT INTO workson (eno, pno, responsibility, hours) VALUES

(10002, 1, 'Coding', 20),

(10003, 1, 'Testing', 15),

(10004, 2, 'Designing', 10),

(10005, 2, 'Documentation', 8),

(10001, 3, 'Management', 40);

1.

1. Query to return departments ordered by ascending department name:

sql Copy code

```sql
SELECT *
FROM dept
ORDER BY dname ASC;
```

2.

2. Query to return employee name, department name, and employee title:

sql Copy code

```sql
SELECT emp.ename, dept.dname, emp.title
FROM emp
JOIN dept ON emp.dno = dept.dno;
```

3.

3. Query to return project name, hours worked, and project number for works on records where hours > 10:

sql Copy code

```sql
SELECT proj.pname, workson.hours, proj.pno
FROM workson
JOIN proj ON workson.pno = proj.pno
WHERE workson.hours > 10;
```

4.

4. Query to return project name, department name, and budget for projects with a budget < Rs. 50,000:

```sql
SELECT proj.pname, dept.dname, proj.budget
FROM proj
JOIN dept ON proj.dno = dept.dno
WHERE proj.budget < 50000;
```

5.

5. Query to find the responsibility of the employee "Ramesh" who is working on project "Banking":

```sql
SELECT workson.responsibility
FROM workson
JOIN emp ON workson.eno = emp.eno
JOIN proj ON workson.pno = proj.pno
WHERE emp.ename = 'Ramesh' AND proj.pname = 'Banking';
```

S8)

```
-- Create the 'emp' table
CREATE TABLE emp (
  eno INT PRIMARY KEY,
  ename VARCHAR(50),
  bdate DATE,
```

```
  title VARCHAR(50),
  salary DECIMAL(10, 2),
  dno INT,
  FOREIGN KEY (dno) REFERENCES dept(dno)
);

-- Create the 'proj' table
CREATE TABLE proj (
  pno INT PRIMARY KEY,
  pname VARCHAR(50),
  budget DECIMAL(10, 2),
  dno INT,
  FOREIGN KEY (dno) REFERENCES dept(dno)
);

-- Create the 'dept' table
CREATE TABLE dept (
  dno INT PRIMARY KEY,
  dname VARCHAR(50),
  mgreno INT
);

-- Create the 'workson' table
CREATE TABLE workson (
  eno INT,
  pno INT,
  resp VARCHAR(50),
  hours INT,
  FOREIGN KEY (eno) REFERENCES emp(eno),
  FOREIGN KEY (pno) REFERENCES proj(pno)
);
```

```
-- Insert values into 'emp' table
INSERT INTO emp (eno, ename, bdate, title, salary, dno)
VALUES
  (1, 'John Doe', '1990-05-10', 'Manager', 5000.00, 1),
  (2, 'Jane Smith', '1995-02-15', 'Consultant', 4000.00, 1),
  (3, 'Michael Johnson', '1988-09-20', 'Analyst', 3000.00, 2),
  (4, 'Emily Brown', '1992-11-30', 'Consultant', 4500.00, 2);

-- Insert values into 'proj' table
INSERT INTO proj (pno, pname, budget, dno)
VALUES
  (1, 'Project A', 10000.00, 1),
  (2, 'Project B', 15000.00, 2);

-- Insert values into 'dept' table
INSERT INTO dept (dno, dname, mgreno)
VALUES
  (1, 'Consulting', 1),
  (2, 'Production', 2);

-- Insert values into 'workson' table
INSERT INTO workson (eno, pno, resp, hours)
VALUES
  (1, 1, 'Lead', 40),
  (2, 1, 'Developer', 30),
  (3, 2, 'Designer', 20),
  (4, 2, 'Tester', 35);
```

1.

1. SQL query to return the employee numbers and salaries of all employees in the 'Consulting' department ordered by descending salary:

sql Copy code

```sql
SELECT emp.eno, emp.salary
FROM emp
JOIN dept ON emp.dno = dept.dno
WHERE dept.dname = 'Consulting'
ORDER BY emp.salary DESC;
```

2.

2. SQL query to return the employee name, project name, employee title, and hours for all 'workson' records:

sql Copy code

```sql
SELECT emp.ename, proj.pname, emp.title, workson.hours
FROM workson
JOIN emp ON workson.eno = emp.eno
JOIN proj ON workson.pno = proj.pno;
```

3.

3. SQL query to find all employees whose salary is between 1000 and 8000:

sql Copy code

```sql
SELECT *
FROM emp
WHERE salary BETWEEN 1000 AND 8000;
```

4.

4. SQL query to list all project names:

```sql
SELECT pname
FROM proj;
```

5.

```
SELECT emp.*
FROM emp
JOIN workson ON emp.eno = workson.eno
JOIN proj ON workson.pno = proj.pno
JOIN dept ON proj.dno = dept.dno
WHERE proj.pname = 'Banking' AND dept.dname = 'Production' AND workson.hours = 120;
```

S9)

```
CREATE TABLE Employee (
  employee_name VARCHAR(50) PRIMARY KEY,
  street VARCHAR(100),
  city VARCHAR(100)
);

CREATE TABLE Company (
  company_name VARCHAR(100) PRIMARY KEY,
  city VARCHAR(100)
);

CREATE TABLE Works (
  employee_name VARCHAR(50),
  company_name VARCHAR(100),
  salary DECIMAL(10, 2),
```

```
  FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
  FOREIGN KEY (company_name) REFERENCES Company(company_name)
);

CREATE TABLE Manages (
  employee_name VARCHAR(50),
  manager_name VARCHAR(50),
  FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
  FOREIGN KEY (manager_name) REFERENCES Employee(employee_name)
);
```

1.

1. Count employees company-wise where salary is greater than 25000:

sql Copy code

```sql
SELECT company_name, COUNT(employee_name) AS total_employees
FROM Works
WHERE salary > 25000
GROUP BY company_name;
```

2.

2. Delete the column "salary" from the Works table:

sql Copy code

```sql
ALTER TABLE Works
DROP COLUMN salary;
```

3,

3. Display the structure of the Manages table:

```sql
DESCRIBE Manages;
```

4.

4. Update the data type of the employee_name column in the Manages table from varchar(30) to varchar(50):

```sql
ALTER TABLE Manages
MODIFY COLUMN employee_name VARCHAR(50);
```

5.

5. Find employees whose salary ranges from 25000 to 50000:

```sql
SELECT employee_name
FROM Works
WHERE salary BETWEEN 25000 AND 50000;
```

S10)

1.

1. Find the names, street address, and cities of residence for all employees:

```sql
SELECT employee_name, street, city
FROM Employee;
```

2.

2. Find the names of all employees in the database who live in the city "Pune":

```sql
SELECT employee_name
FROM Employee
WHERE city = 'Pune';
```

3.

3. Find the names of all employees in the database who do not work for 'First Bank Corporation':

```sql
SELECT employee_name
FROM Employee
WHERE employee_name NOT IN (
  SELECT employee_name
  FROM Works
  WHERE company_name = 'First Bank Corporation'
);
```

5.

5. Find all the managers:

```sql
SELECT DISTINCT manager_name
FROM Manages;
```

4.

4. Find the names of all employees in the database who earn more than every employee of 'Small Bank Corporation':

sql Copy code

```sql
SELECT employee_name
FROM Employee
WHERE employee_name IN (
  SELECT employee_name
  FROM Works
  WHERE salary > (
    SELECT MAX(salary)
    FROM Works
    WHERE company_name = 'Small Bank Corporation'
  )
);
```

S11)

-- 1) Create the Employee table

CREATE TABLE Employee (

EmployerId INT,

Lastname VARCHAR(50),

Firstname VARCHAR(50),

Middlename VARCHAR(50),

JobId INT,

ManagerId INT,

Hiredate DATE,

Salary INT,

DepartmentId INT

);

-- 2) Insert the records into the Employee table

INSERT INTO Employee (EmployerId, Lastname, Firstname, Middlename, JobId, ManagerId, Hiredate, Salary, DepartmentId)

VALUES

(7369, 'Smith', 'Jon', 'Q', 667, 7902, '2021-12-17', 800, 10),

(7499, 'Allen', 'Kevin', '1', 670, 7698, '2022-02-20', 1600, 20),

(7505, 'Doyle', 'Jean', 'K', 671, 7839, '2023-04-04', 2850, 20),

(7505, 'Doyle', 'Lynn', 'S', 671, 7839, '2023-05-15', 2750, 30),

(7505, 'Doyle', 'Leslie', 'D', 671, 7839, '2023-06-10', 2200, 40),

(7505, 'Doyle', 'Cynthia', 'D', 670, 7698, '2022-02-22', 1250, 10),

(7506, 'Dennis', 'Baker', NULL, 7521, NULL, NULL, NULL, NULL);

1)

1. Create a view for all columns of the Employee table:

sql Copy code

```sql
CREATE VIEW Employee_View AS
SELECT * FROM Employee;
```

2.

2. Create a view of last name, first name, and middle name of the Employee table:

sql Copy code

```sql
CREATE VIEW Name_View AS
SELECT Lastname, Firstname, Middlename FROM Employee;
```

3.

3. Create a view of all employees whose last name starts with "S" and middle name is "Q":

sql Copy code

```sql
CREATE VIEW SQ_Employees_View AS
SELECT * FROM Employee
WHERE Lastname LIKE 'S%' AND Middlename = 'Q';
```

4.

CREATE VIEW Salary_Increment_View AS

SELECT EmployerId, Lastname, Firstname, Middlename, JobJd, ManagerJd, Hiredate, Salar * 1.1 AS Salar, Department_id

FROM Employee;

5.

5. To delete the views created in the previous steps:

sql Copy code

```sql
DROP VIEW Employee_View;
DROP VIEW Name_View;
DROP VIEW SQ_Employees_View;
DROP VIEW Salary_Increment_View;
```

S12)

1. CREATE TABLE Animal (

id INT AUTO_INCREMENT,

name VARCHAR(50),

PRIMARY KEY (id)

);

INSERT INTO Animal (name) VALUES ('Cat'), ('Dog'), ('Elephant'), ('Lion');

2.

2. Create the table Location with an auto-increment field starting with 100:

sql Copy code

```sql
CREATE TABLE Location (
  LocationId INT AUTO_INCREMENT PRIMARY KEY,
  RegionalGroup VARCHAR(50)
) AUTO_INCREMENT = 100;
```

3.

3. Rename the Location table to "Location of India" and display its content:

```sql
ALTER TABLE Location RENAME TO `Location of India`;

SELECT * FROM `Location of India`;
```

4.

4. Create a view of all locations where the LocationId is 101:

```sql
CREATE VIEW Location101 AS
SELECT * FROM `Location of India` WHERE LocationId = 101;
```

5.

5. Alter the table Location to add a column named "LocationName":

```sql
ALTER TABLE `Location of India` ADD COLUMN LocationName VARCHAR(50);
```

S13)

Location (Location_Id, Regional_Group)

Department (Department_Id, Name, Location_Id)

Job (Job_Id, Function)

Employee (Employee_Id, Lastname, Firstname, Middlename, Job_Id, Manager_Id, Hiredate, Salary, Department_Id)

1.

1. List the details about "smith":

```sql
SELECT * FROM Employee WHERE Lastname = 'Smith';
```

2.

2. List of the employees whose job id is 671:

```sql
SELECT * FROM Employee WHERE Job_Id = 671;
```

3.

3. List out the employees who are earning a salary between 3000 and 4500:

```sql
SELECT * FROM Employee WHERE Salary BETWEEN 3000 AND 4500;
```

4.

4. List out the employees who are working in department 10 or 20:

```sql
SELECT * FROM Employee WHERE Department_Id IN (10, 20);
```

5.

5. Find out the employees who are not working in department 10 or 30:

```sql
SELECT * FROM Employee WHERE Department_Id NOT IN (10, 30);
```

S14)

```
-- Create 'product' table
CREATE TABLE product (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(255),
  supplier_name VARCHAR(255),
  unit_price DECIMAL(10, 2)
);

-- Create 'product_price_history' table
CREATE TABLE product_price_history (
  product_id INT,
  product_name VARCHAR(255),
  supplier_name VARCHAR(255),
  unit_price DECIMAL(10, 2),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create trigger to update 'product_price_history' table
CREATE TRIGGER update_price_history
AFTER UPDATE ON product
FOR EACH ROW
BEGIN
  INSERT INTO product_price_history (product_id, product_name, supplier_name, unit_price)
  VALUES (NEW.product_id, NEW.product_name, NEW.supplier_name, NEW.unit_price);
END;
```

S15)

create a table called "account" with columns "accno" and "amount" in SQL:

```sql
CREATE TABLE account (
  accno INT,
  amount INT
);
```

create a trigger on the "account" table that will check the new inserted amount before an update and perform certain actions based on its value:

```
CREATE TRIGGER before_account_update
BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
 IF NEW.amount < 0 THEN
  SET NEW.amount = 0;
 ELSEIF NEW.amount > 100 THEN
  SET NEW.amount = 100;
 END IF;
END;
```

S16)

Procedure to find the maximum number from given three numbers:

```
PROCEDURE FindMaximumNumber(num1 INT, num2 INT, num3 INT)
BEGIN
  DECLARE maxNum INT;

  IF num1 >= num2 AND num1 >= num3 THEN
    SET maxNum = num1;
  ELSEIF num2 >= num1 AND num2 >= num3 THEN
    SET maxNum = num2;
  ELSE
```

```
        SET maxNum = num3;
    END IF;

    SELECT maxNum AS MaximumNumber;
END;
```

create a table and insert values:

```
CREATE TABLE studentdata (
    mark1 INT,
    mark2 INT,
    mark3 INT,
    mark4 INT,
    student_name VARCHAR(50)
);

INSERT INTO studentdata (mark1, mark2, mark3, mark4, student_name)
VALUES
    (90, 85, 95, 92, 'John Doe'),
    (78, 82, 80, 85, 'Jane Smith'),
    (88, 90, 92, 87, 'Alice Johnson');
```

To list the average marks of each student, you can use the following query:

```
SELECT student_name, CalculateAverageMarks(mark1, mark2, mark3, mark4) AS average_marks
FROM studentdata;
```

Function to calculate the average of marks:

```sql
CREATE FUNCTION CalculateAverageMarks(
    mark1 INT,
    mark2 INT,
    mark3 INT,
    mark4 INT
) RETURNS FLOAT
BEGIN
    DECLARE average FLOAT;

    SET average = (mark1 + mark2 + mark3 + mark4) / 4.0;

    RETURN average;
END;
```

S17)

```
-- Create a table to store student information
CREATE TABLE students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50),
  marks INT
);


-- Insert sample data
INSERT INTO students (name, marks) VALUES
  ('John', 85),
  ('Alice', 92),
  ('Bob', 78),
  ('Emily', 68),
  ('David', 95),
```

```
  ('Sarah', 87),
  ('Michael', 72),
  ('Emma', 90),
  ('Daniel', 83),
  ('Olivia', 79);

-- Create a procedure to calculate the grade
DELIMITER //

CREATE PROCEDURE calculate_grade()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE student_id INT;
  DECLARE student_name VARCHAR(50);
  DECLARE student_marks INT;
  DECLARE cur CURSOR FOR SELECT id, name, marks FROM students;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;

  read_loop: LOOP
    FETCH cur INTO student_id, student_name, student_marks;
    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Calculate the grade based on marks
    DECLARE student_grade CHAR(1);
    IF student_marks >= 90 THEN
      SET student_grade = 'A';
    ELSEIF student_marks >= 80 THEN
```

```
    SET student_grade = 'B';
   ELSEIF student_marks >= 70 THEN
    SET student_grade = 'C';
   ELSEIF student_marks >= 60 THEN
    SET student_grade = 'D';
   ELSE
    SET student_grade = 'F';
   END IF;

   -- Display the grade for each student
   SELECT CONCAT('Student Name: ', student_name, ', Grade: ', student_grade) AS result;
  END LOOP;

  CLOSE cur;
END //

DELIMITER ;

-- Call the procedure to calculate grades
CALL calculate_grade();
```

S18)

```
-- Create the student table
CREATE TABLE student (
  RollNo INT,
  mark1 INT,
  mark2 INT,
  mark3 INT,
  mark4 INT
);
```

```
-- Insert sample values
INSERT INTO student (RollNo, mark1, mark2, mark3, mark4)
VALUES (1, 80, 90, 75, 85),
       (2, 70, 65, 80, 90),
       (3, 85, 95, 70, 80);

-- Create the procedure
DELIMITER //
CREATE PROCEDURE CalculateAverage()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE rollNo INT;
  DECLARE m1, m2, m3, m4 INT;
  DECLARE cur CURSOR FOR SELECT RollNo, mark1, mark2, mark3, mark4 FROM student;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;

  read_loop: LOOP
    FETCH cur INTO rollNo, m1, m2, m3, m4;
    IF done = 1 THEN
      LEAVE read_loop;
    END IF;

    -- Calculate average marks
    DECLARE average DECIMAL(5,2);
    SET average = (m1 + m2 + m3 + m4) / 4;

    -- Display the result
    SELECT CONCAT('Roll No: ', rollNo, ', Average Marks: ', average) AS Result;
  END LOOP;
```

```
  CLOSE cur;
END //
DELIMITER ;

-- Call the procedure
CALL CalculateAverage();
```

S20)

1. Create a stored procedure:

```
DELIMITER //

CREATE PROCEDURE FindOddEven(IN num INT)
BEGIN
    DECLARE result VARCHAR(10);

    IF num % 2 = 0 THEN
        SET result = 'Even';
    ELSE
        SET result = 'Odd';
    END IF;

    SELECT result AS `Number Type`;
END //

DELIMITER ;
```

2. Call the stored procedure:

```sql
CALL FindOddEven(10);
```

This will return 'Even' because 10 is an even number.

3. Create a function:

```
DELIMITER //

CREATE FUNCTION IsEven(num INT) RETURNS VARCHAR(10)
BEGIN
    DECLARE result VARCHAR(10);

    IF num % 2 = 0 THEN
        SET result = 'Even';
    ELSE
        SET result = 'Odd';
    END IF;

    RETURN result;
END //

DELIMITER ;
```

4. Call the function:

```sql
SELECT IsEven(7);
```

This will return 'Odd' because 7 is an odd number.