❖EXERCISE 3:

1) Write a program containing a loop that iterates from 1 to 1000 using a variable I, which is incremented each time around the loop. The program should output the value of I every hundred iterations (i.e., the output should be 100, 200, etc.).

⇒delimiter ;

```
CREATE PROCEDURE OutputEveryHundred()
   -> BEGIN
   -> DECLARE i INT DEFAULT 1;
   -> WHILE i <= 1000 DO
   -> IF i % 100 = 0 THEN
   -> SELECT i AS Output;
-> END IF;
   -> SET i = i + 1;
   -> END WHILE;
   -> END //
delimiter ;
CALL OutputEveryHundred();
```

2) Write a program that examines all the numbers from 1 to 999, displaying all those for which the sum of the cubes of the digits equal the number itself.

⇒ DELIMITER //

```
CREATE PROCEDURE FindArmstrongNumbers()
BEGIN
DECLARE num INT;
DECLARE digit INT;
DECLARE sum_of_cubes INT;
```

```
    DECLARE temp INT;
    SET num = 1;
    WHILE num < 1000 DO
        SET temp = num;
        SET sum_of_cubes = 0;
        WHILE temp > 0 DO
            SET digit = temp % 10;              -- Get the last digit
            SET sum_of_cubes = sum_of_cubes + (digit * digit * digit);
SET temp = temp DIV 10;              -- Remove the last digit
        END WHILE;
        IF sum_of_cubes = num THEN
            SELECT num AS ArmstrongNumber;
        END IF;
SET num = num + 1;
    END WHILE;
END //
DELIMITER ;
CALL FindArmstrongNumbers();
```

3) Write a program that Selects from any table a minimum and maximum value for a radius, along with an increment factor, and generates a series of radii by repeatedly adding the increment to the minimum until the maximum is reached. For each value of the radius, compute and display the circumference, area, and volume of the sphere. (Be sure to include both the maximum and the minimum values.)

⇒ DELIMITER //

```
CREATE PROCEDURE GenerateRadiusSeries()
BEGIN
    DECLARE min_radius DECIMAL(10,2);
    DECLARE max_radius DECIMAL(10,2);
    DECLARE increment DECIMAL(10,2);
    DECLARE current_radius DECIMAL(10,2);
    SELECT num, square, `cube` INTO min_radius, max_radius, increment
    FROM tempp
    LIMIT 1;

    SET current_radius = min_radius;

       WHILE current_radius <= max_radius DO
        SET @circumference = 2 * PI() * current_radius;
        SET @area = 4 * PI() * current_radius * current_radius;
        SET @volume = (4/3) * PI() * current_radius * current_radius * current_radius;

            SELECT
          current_radius AS Radius,
          @circumference AS Circumference,
          @area AS Area,
          @volume AS Volume;
```

```
        SET current_radius = current_radius + increment;
    END WHILE;
END //


DELIMITER ;
CALL GenerateRadiusSeries();
```

4) A palindrome is a word that is spelled the same forward and backward, such as level, radar, etc. Write a program to Selects from any table a five letter word and determine whether it is a palindrome.

⇒

```
DELIMITER //
CREATE PROCEDURE CheckPalindrome()
BEGIN
    DECLARE selected_word CHAR(5);
    DECLARE reversed_word CHAR(5);
    SELECT Sname INTO selected_word FROM salespeople WHERE CHAR_LENGTH(Sname) = 5 LIMIT 1;
    SET reversed_word = REVERSE(selected_word);
    IF selected_word = reversed_word THEN
        SELECT selected_word AS Word, 'is a palindrome.' AS Result;
    ELSE
        SELECT selected_word AS Word, 'is not a palindrome.' AS Result;
    END IF;
END //


DELIMITER ;
```

```
CALL CheckPalindrome();
```

5) Modify the above program to Select from any table a variable length word. This requires determining how many characters are read in.

⇒

```
DELIMITER //
CREATE PROCEDURE CheckVariableLengthPalindrome()
BEGIN
   DECLARE selected_word VARCHAR(255);
   DECLARE reversed_word VARCHAR(255);
   DECLARE word_length INT;
   SELECT Sname INTO selected_word FROM salespeople LIMIT 1;
   SET word_length = CHAR_LENGTH(selected_word);
   SET reversed_word = REVERSE(selected_word);
   IF selected_word = reversed_word THEN
      SELECT selected_word AS Word, word_length AS Length, 'is a palindrome.' AS Result;
   ELSE
      SELECT selected_word AS Word, word_length AS Length, 'is not a palindrome.' AS Result;
   END IF;
END //

DELIMITER ;
CALL CheckVariableLengthPalindrome();
```