Name: Samiksha Bhashte

Roll No: C05

Assignment-6

```
CREATE TABLE Course (
  course_num INTEGER PRIMARY KEY,
  course_name VARCHAR2(20),
  dept_name VARCHAR2(15),
  credits INTEGER
);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
  (1, 'Cpp', 'CSE', 4);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
  (2, 'Data Structures', 'CSE', 3);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
  (3, 'Computational Mathematics', 'MATH', 4);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
  (4, 'C Programming', 'CSE', 3);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
  (5, 'Database Management', 'CSE', 4);

INSERT INTO Course (course_num, course_name, dept_name, credits)
VALUES
```

```
(6, 'Physics', 'PHYSICS', 4);

SELECT
    * FROM
    Course;
```

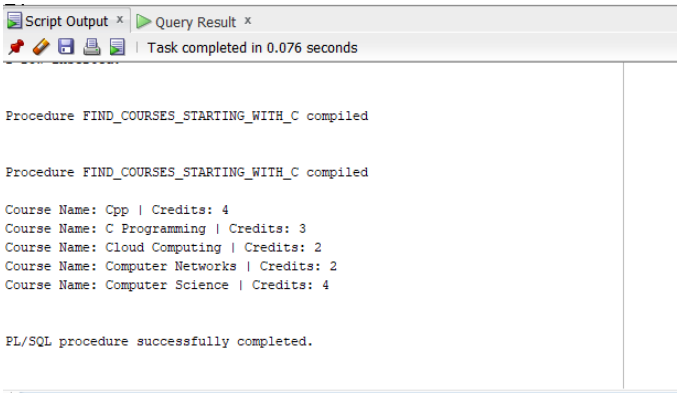
Script Output × | Query Result ×

SQL | All Rows Fetched: 8 in 0.006 seconds

	COURSE_NUM	COURSE_NAME	DEPT_NAME	CREDITS
1	1	Cpp	CSE	4
2	2	Data Structures	CSE	3
3	4	C Programming	CSE	3
4	5	Database Management	CSE	4
5	6	Physics	PHYSICS	4
6	7	Cloud Computing	CSE	2
7	8	Computer Networks	CSE	2
8	9	Computer Science	CSE	4

```
-- Create the procedure to find courses starting with 'C'
CREATE OR REPLACE PROCEDURE find_courses_starting_with_C AS
  -- Define a cursor to fetch course_name and credits where course_name starts with 'C'
  CURSOR course_cursor IS
    SELECT course_name, credits
    FROM Course
    WHERE UPPER(course_name) LIKE 'C%';

  -- Declare variables to hold the fetched data
  v_course_name Course.course_name%TYPE;
  v_credits     Course.credits%TYPE;

BEGIN
  -- Open the cursor
  OPEN course_cursor;

  -- Fetch data row by row
  LOOP
    FETCH course_cursor INTO v_course_name, v_credits;
    EXIT WHEN course_cursor%NOTFOUND;

    -- Display the course_name and credits
    DBMS_OUTPUT.PUT_LINE('Course Name: ' || v_course_name || ' | Credits: ' || v_credits);
```

```
  END LOOP;

  -- Close the cursor
  CLOSE course_cursor;
END;
/

-- Execute the procedure and show the output
SET SERVEROUTPUT ON;

BEGIN
  find_courses_starting_with_C;
END;
/
```



---q2 Write a procedure which includes cursors: Find course names from ‘CSE’ department

```
CREATE OR REPLACE PROCEDURE find_courses_from_cse AS
  -- Define a cursor to fetch course_name from Course where dept_name is 'CSE'
  CURSOR course_cursor IS
    SELECT course_name
    FROM Course
    WHERE dept_name = 'CSE';

  -- Declare a variable to hold the fetched data
  v_course_name Course.course_name%TYPE;
```

```
BEGIN
  -- Open the cursor
  OPEN course_cursor;

  -- Fetch data row by row
  LOOP
    FETCH course_cursor INTO v_course_name;
    EXIT WHEN course_cursor%NOTFOUND;

    -- Display the course_name
    DBMS_OUTPUT.PUT_LINE('Course Name: ' || v_course_name);
  END LOOP;

  -- Close the cursor
  CLOSE course_cursor;
END;
/

-- Execute the procedure and show the output
SET SERVEROUTPUT ON;

BEGIN
  find_courses_from_cse;
END;
/
```

Script Output × Query Result ×

Task completed in 0.081 seconds

```
PL/SQL procedure successfully completed.


Procedure FIND_COURSES_FROM_CSE compiled

Course Name: Cpp
Course Name: Data Structures
Course Name: C Programming
Course Name: Database Management
Course Name: Cloud Computing
Course Name: Computer Networks
Course Name: Computer Science


PL/SQL procedure successfully completed.
```

--Q. 2 create a stored procedure that uses a cursor to retrieve data from a table, processes each row, and inserts the processed data into another table.

---Question 2

```
CREATE TABLE orders (
    order_id   NUMBER PRIMARY KEY,
    cust_name  VARCHAR2(50)
);

CREATE TABLE processed_orders (
    order_id   NUMBER PRIMARY KEY,
    cust_name  VARCHAR2(50),
    status     VARCHAR2(20)
);

INSERT INTO orders (order_id, cust_name) VALUES (1, 'John Doe');
INSERT INTO orders (order_id, cust_name) VALUES (2, 'Jane Smith');
INSERT INTO orders (order_id, cust_name) VALUES (3, 'Alice Johnson');

SELECT * FROM orders;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 3 in 0.006 seconds

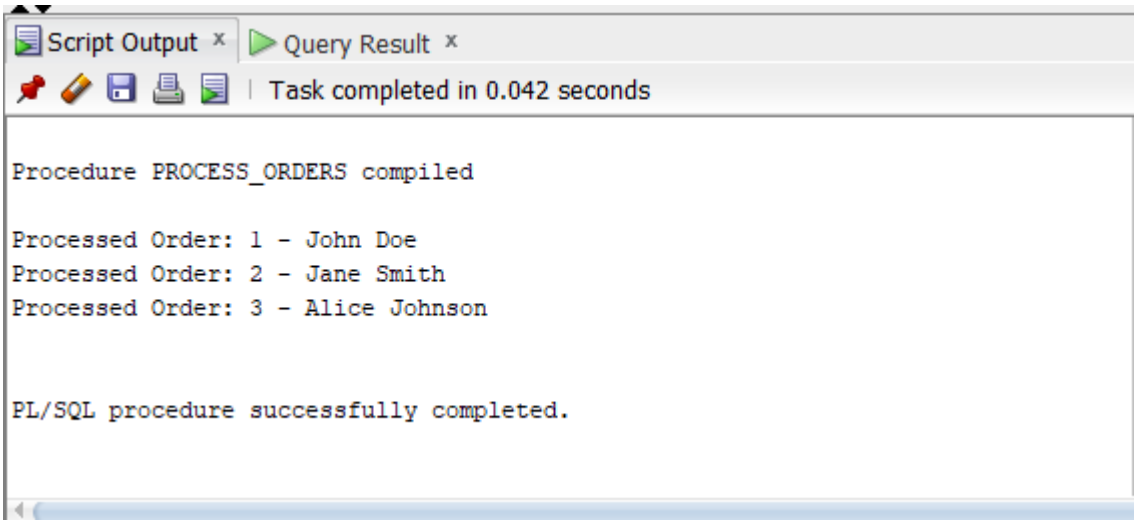
	ORDER_ID	CUST_NAME
1	1	John Doe
2	2	Jane Smith
3	3	Alice Johnson

```
-- Create the orders table
CREATE TABLE orders (
  order_id   NUMBER PRIMARY KEY,
  cust_name  VARCHAR2(50)
);

-- Create the processed_orders table
CREATE TABLE processed_orders (
  order_id   NUMBER PRIMARY KEY,
  cust_name  VARCHAR2(50),
  status     VARCHAR2(20)
);

-- Insert data into the orders table
INSERT INTO orders (order_id, cust_name) VALUES (1, 'John Doe');
INSERT INTO orders (order_id, cust_name) VALUES (2, 'Jane Smith');
INSERT INTO orders (order_id, cust_name) VALUES (3, 'Alice Johnson');
```

1. ---q1Use a cursor to loop through each row in the orders table.

```
-- Create the procedure to process orders
CREATE OR REPLACE PROCEDURE process_orders
IS
  -- Define a cursor to retrieve all rows from the orders table
  CURSOR order_cursor IS
    SELECT order_id, cust_name
    FROM orders;

  -- Declare variables to hold each row’s data
  v_order_id   orders.order_id%TYPE;
  v_cust_name  orders.cust_name%TYPE;

BEGIN
  -- Open the cursor
  OPEN order_cursor;

  -- Loop through each row in the orders table
  LOOP
    -- Fetch data into variables
    FETCH order_cursor INTO v_order_id, v_cust_name;

    -- Exit the loop when no more rows are found
    EXIT WHEN order_cursor%NOTFOUND;

    -- Insert the processed order into the processed_orders table
    INSERT INTO processed_orders (order_id, cust_name, status)
    VALUES (v_order_id, v_cust_name, 'Processed');
```

```
    -- Display the processed order info (optional, useful for debugging)
    DBMS_OUTPUT.PUT_LINE('Processed Order: ' || v_order_id || ' - ' || v_cust_name);
  END LOOP;

  -- Close the cursor
  CLOSE order_cursor;

  -- Commit the transaction
  COMMIT;
END;
/

-- Execute the procedure
BEGIN
  process_orders;
END;
/
```



--q2For each order, update its status to 'Processed' and insert it into the processed_orders table.

- --Show the output

SET SERVEROUTPUT ON;

-- Display the processed orders

SELECT * FROM processed_orders;

Script Output × Query Result ×

SQL | All Rows Fetched: 3 in 0.013 seconds

	ORDER_ID	CUST_NAME	STATUS
1	1	John Doe	Processed
2	2	Jane Smith	Processed
3	3	Alice Johnson	Processed

--Q3 Create a stored procedure that uses a cursor to fetch records from one table (students), processes each record by applying a logic (marking whether the student passed or failed based on their grade), and inserts the processed records into another table (processed_students).

```
CREATE TABLE students (
  student_id NUMBER PRIMARY KEY,
  name VARCHAR2(50),
  grades NUMBER
);

CREATE TABLE processed_students (
  student_id NUMBER PRIMARY KEY,
  name VARCHAR2(50),
  grades NUMBER,
  result VARCHAR2(10)
);
INSERT INTO students (student_id, name, grades) VALUES (1, 'John Doe', 85);
INSERT INTO students (student_id, name, grades) VALUES (2, 'Jane Smith', 60);
INSERT INTO students (student_id, name, grades) VALUES (3, 'Alice Johnson', 90);
```

--Q1Students (student_id, name, grades): Holds information about students and their grades.

```
CREATE OR REPLACE PROCEDURE process_students AS
  -- Define a cursor to fetch records from the students table
  CURSOR student_cursor IS
    SELECT student_id, name, grades
    FROM students;

  -- Declare variables to hold each record's data
  v_student_id students.student_id%TYPE;
  v_name students.name%TYPE;
  v_grades students.grades%TYPE;
  v_result processed_students.result%TYPE;
```

```
BEGIN
  -- Open the cursor
  OPEN student_cursor;

  -- Loop through each record in the students table
  LOOP
    -- Fetch data into variables
    FETCH student_cursor INTO v_student_id, v_name, v_grades;

    -- Exit the loop when no more records are found
    EXIT WHEN student_cursor%NOTFOUND;

    -- Process each record by applying the logic
    IF v_grades >= 60 THEN
      v_result := 'Passed';
    ELSE
      v_result := 'Failed';
    END IF;

    -- Insert the processed record into the processed_students table
    INSERT INTO processed_students (student_id, name, grades, result)
    VALUES (v_student_id, v_name, v_grades, v_result);

    -- Display the processed record info (optional, useful for debugging)
    DBMS_OUTPUT.PUT_LINE('Processed Student: ' || v_student_id || ' - ' || v_name || ' - ' || v_grades ||
' - ' || v_result);
  END LOOP;

  -- Close the cursor
  CLOSE student_cursor;

  -- Commit the transaction
  COMMIT;
END;
/
```

```
BEGIN
  process_students;
END;
/
```

Script Output | Query Result

Task completed in 0.036 seconds

```
Processed Student: 1 - John Doe - 85 - Passed
Processed Student: 2 - Jane Smith - 60 - Passed
Processed Student: 3 - Alice Johnson - 90 - Passed


PL/SQL procedure successfully completed.
```

--Q2 Processed_students (student_id, name, grades, result): Holds information about whether the student passed or failed.

```
SET SERVEROUTPUT ON;
```

```
SELECT * FROM processed_students;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 3 in 0.012 seconds

	STUDENT_ID	NAME	GRADES	RESULT
1	1	John Doe	85	Passed
2	2	Jane Smith	60	Passed
3	3	Alice Johnson	90	Passed