Branch: MCA (Data Science)	**Semester:** 2nd
Student Name: Chaitanya Sharma	**UID:** 25MCD10056
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: 25MCD- KAR1	**Date of Performance:** 03-02-2026

Experiment No.: 4

Implementation of Iterative Control Structures using FOR, WHILE, and LOOP in PostgreSQL

1. **Aim of the practical:**
   To understand and implement iterative control structures in PostgreSQL conceptually, including FOR loops, WHILE loops, and basic LOOP constructs, for repeated execution of database logic.

2. **Tools Used:** PostgreSQL

3. **Objectives of the practical:**
   - To understand why iteration is required in database programming
   - To learn the purpose and behavior of FOR, WHILE, and LOOP constructs
   - To understand how repeated data processing is handled in databases
   - To relate loop concepts to real-world batch processing scenarios
   - To strengthen conceptual knowledge of procedural SQL used in enterprise systems

4. **Theory:**
   In real-world database applications, tasks often need to be repeated multiple times. Examples include processing employee records, generating reports, validating data, applying salary increments, and running batch jobs. Standard SQL is declarative and works well for single operations, but repeated logic requires procedural control.
   PostgreSQL provides PL/pgSQL, a procedural extension that supports iteration using loop structures. These loops allow SQL statements to execute repeatedly until a specific condition is met.
   Iteration in PostgreSQL is commonly used inside:
   - Stored procedures
   - Functions
   - Anonymous execution blocks

   Large organizations such as Amazon, SAP, Oracle, and Rippling use loop-based logic for payroll processing, billing cycles, analytics, and automation workflows.

   Types of Loops in PostgreSQL

   1. FOR Loop (Range-Based)
   - Executes a fixed number of times
   - Useful when the number of iterations is known in advance
   - Commonly used for counters, testing, and batch execution

2. FOR Loop (Query-Based)
- Iterates over rows returned by a query
- Processes one row at a time
- Frequently used for reporting, audits, and row-wise calculations

3. WHILE Loop
- Executes repeatedly as long as a condition remains true
- Suitable for condition-controlled execution
- Often used in retry logic or threshold-based processing

4. LOOP with EXIT Condition
- Executes indefinitely until explicitly stopped
- Provides maximum control over execution flow
- Used in complex workflows where exit conditions are custom-defined

5. Experiment / Practical Steps:

Prerequisite Understanding

Example 1: FOR Loop – Simple Iteration
- The loop runs a fixed number of times
- Each iteration represents one execution cycle
- Useful for understanding basic loop behavior

Application: Counters, repeated tasks, batch execution

Example 2: FOR Loop with Query (Row-by-Row Processing)
- The loop processes database records one at a time
- Each iteration handles a single row
- Simulates cursor-based processing

Application: Employee reports, audits, data verification

Example 3: WHILE Loop – Conditional Iteration
- The loop runs until a condition becomes false
- Execution depends entirely on the condition
- The condition is checked before every iteration

Application: Retry mechanisms, validation loops

Example 4: LOOP with EXIT WHEN
- The loop does not stop automatically
- An explicit exit condition controls termination
- Gives flexibility in complex logic

Application: Workflow engines, complex decision cycles

Example 5: Salary Increment Using FOR Loop
- Employee records are processed one by one
- Salary values are updated iteratively
- Represents real-world payroll processing

Application: Payroll systems, bulk updates

Example 6: Combining LOOP with IF Condition

- Loop processes each record
- Conditional logic classifies data during iteration
- Demonstrates decision-making inside loops

Application: Employee grading, alerts, categorization logic

6. **Code:**

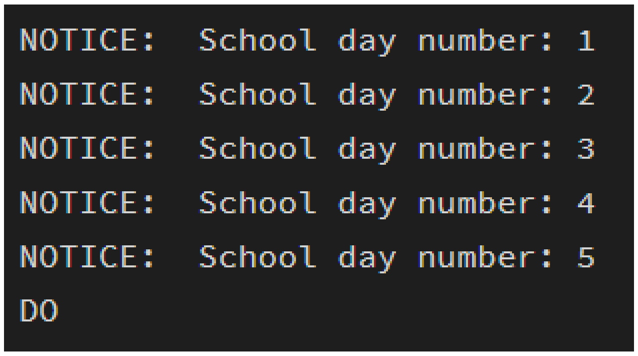
```
-- create table
CREATE TABLE student_registry (
   student_id SERIAL PRIMARY KEY,
   student_name VARCHAR(50) NOT NULL,
   class_name VARCHAR(30) NOT NULL,
   marks INT CHECK (marks >= 0 AND marks <= 100)
);


-- insert sample records
INSERT INTO student_registry (student_name, class_name, marks) VALUES
('Aarav', 'Class 10A', 45),
('Diya', 'Class 10B', 60),
('Kabir', 'Class 10A', 75),
('Meera', 'Class 10C', 52),
('Rohan', 'Class 10B', 40),
('Ishita', 'Class 10A', 88),
('Dev', 'Class 10C', 67),
('Sneha', 'Class 10B', 91);
--   Example 1
  -- FOR Loop – Simple iteration of school days

DO $$
DECLARE
   day_no INT;
BEGIN
   FOR day_no IN 1..5 LOOP
      RAISE NOTICE 'School day number: %', day_no;
   END LOOP;
END $$;
```

```
  --Example 2
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT student_id, student_name FROM student_registry LOOP
      RAISE NOTICE 'Student ID: %, Name: %', rec.student_id, rec.student_name;
   END LOOP;
END $$;



--   Example 3
--   WHILE Loop – Repeated attendance counter
DO $$
DECLARE
   counter INT := 1;
BEGIN
   WHILE counter <= 5 LOOP
      RAISE NOTICE 'Attendance check: %', counter;
      counter := counter + 1;
   END LOOP;
END $$;

  --Example 4
 --  LOOP with EXIT WHEN – Controlled repetition

DO $$
DECLARE
   period_no INT := 1;
BEGIN
   LOOP
      RAISE NOTICE 'Class period: %', period_no;
      period_no := period_no + 1;

      EXIT WHEN period_no > 5;
   END LOOP;
END $$;
--   Example 5
 --  Increase marks for every student (bonus marks)

DO $$
DECLARE
   rec RECORD;
```

```
BEGIN
   FOR rec IN SELECT student_id FROM student_registry LOOP
      UPDATE student_registry
      SET marks = marks + 5   -- bonus marks
      WHERE student_id = rec.student_id;
   END LOOP;
END $$;

SELECT * FROM student_registry;

--   Example 6
  -- Classify students based on marks

DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT student_name, marks FROM student_registry LOOP
      IF rec.marks >= 60 THEN
         RAISE NOTICE '% is a High Performer', rec.student_name;
      ELSE
         RAISE NOTICE '% is an Average Performer', rec.student_name;
      END IF;
   END LOOP;
END $$;
```

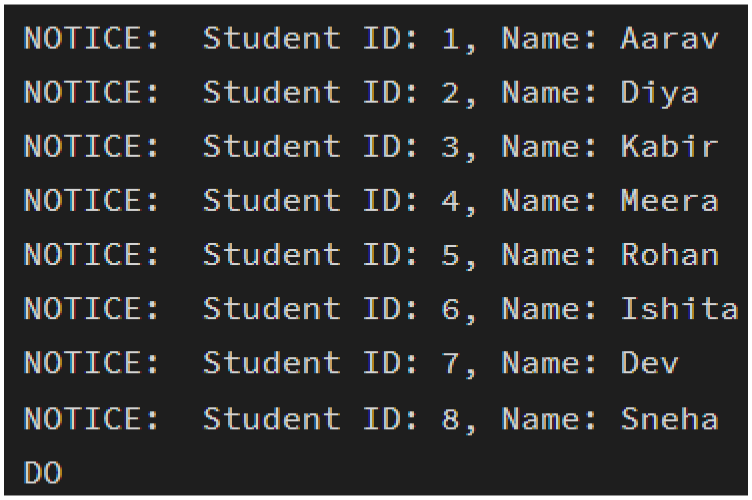
7. **Output:**

Example 1:



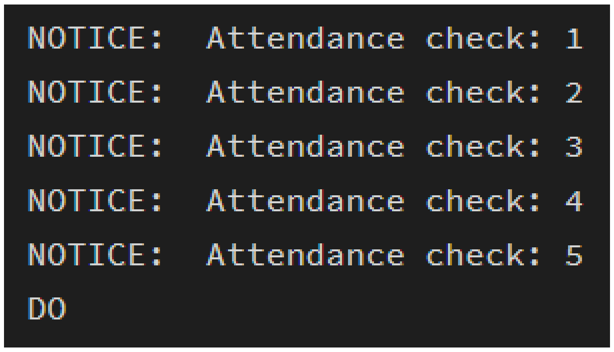
```
NOTICE:  School day number: 1
NOTICE:  School day number: 2
NOTICE:  School day number: 3
NOTICE:  School day number: 4
NOTICE:  School day number: 5
DO
```

Example 2:



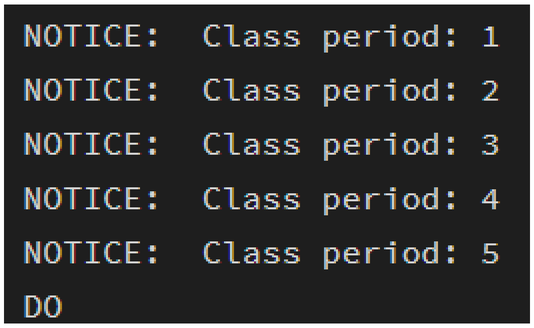
```
NOTICE:  Student ID: 1, Name: Aarav
NOTICE:  Student ID: 2, Name: Diya
NOTICE:  Student ID: 3, Name: Kabir
NOTICE:  Student ID: 4, Name: Meera
NOTICE:  Student ID: 5, Name: Rohan
NOTICE:  Student ID: 6, Name: Ishita
NOTICE:  Student ID: 7, Name: Dev
NOTICE:  Student ID: 8, Name: Sneha
DO
```

Example 3:



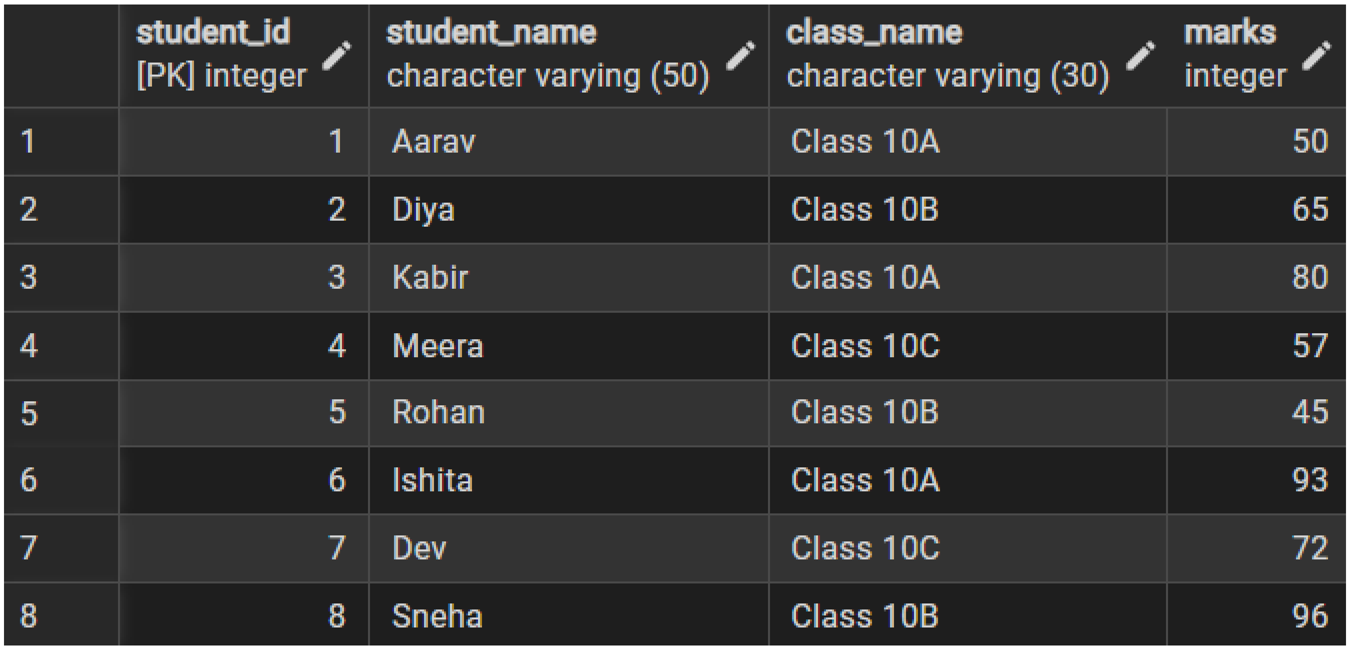
```
NOTICE:  Attendance check: 1
NOTICE:  Attendance check: 2
NOTICE:  Attendance check: 3
NOTICE:  Attendance check: 4
NOTICE:  Attendance check: 5
DO
```

Example 4:



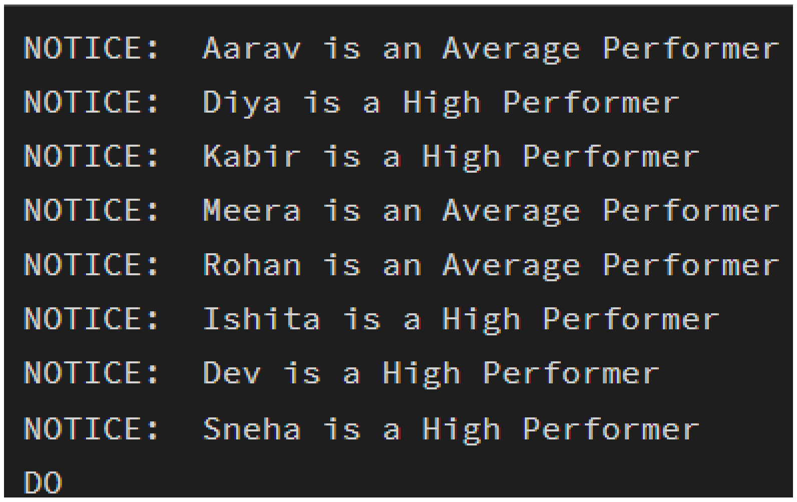
```
NOTICE:  Class period: 1
NOTICE:  Class period: 2
NOTICE:  Class period: 3
NOTICE:  Class period: 4
NOTICE:  Class period: 5
DO
```

Example 5:



	student_id [PK] integer	student_name character varying (50)	class_name character varying (30)	marks integer
1	1	Aarav	Class 10A	50
2	2	Diya	Class 10B	65
3	3	Kabir	Class 10A	80
4	4	Meera	Class 10C	57
5	5	Rohan	Class 10B	45
6	6	Ishita	Class 10A	93
7	7	Dev	Class 10C	72
8	8	Sneha	Class 10B	96

Example 6:



```
NOTICE:  Aarav is an Average Performer
NOTICE:  Diya is a High Performer
NOTICE:  Kabir is a High Performer
NOTICE:  Meera is an Average Performer
NOTICE:  Rohan is an Average Performer
NOTICE:  Ishita is a High Performer
NOTICE:  Dev is a High Performer
NOTICE:  Sneha is a High Performer
DO
```

8. Learning Outcomes:

- Developed a clear understanding of iteration mechanisms within PostgreSQL
- Recognized when to use FOR, WHILE, and LOOP control statements effectively
- Practiced building procedural workflows using PL/pgSQL looping constructs
- Utilized loops to process records sequentially and apply conditional logic
- Built the fundamental skills necessary for designing enterprise-grade database solutions