SE Comp A	Roll number : 9913
Experiment no. : 8	Date of Implementation : 26/3/2024
Aim : To implement PL/pgSQL	
Tool Used : PostgreSQL	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 8	PL/pgSQL

	Aim	To implement PL/pgSQL
	Tools	PostgreSQL http://www.postgresqltutorial.com/postgresql-stored-procedures/ mysql https://dev.mysql.com/doc/refman/8.0/en/cursors.html https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/ https://dev.mysql.com/doc/refman/8.0/en/error-message-elements.html https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx
	Procedure	PL/pgSQL is a loadable procedural language for the Postgres database system.

This package was originally written by Jan Wieck. The design goals of PL/pgSQL were to create a loadable procedural language that can be used to create functions and trigger procedures, adds control structures to the SQL language.

Structure of PL/pgSQL

PL/pgSQL is a block-structured language. The complete text of a function definition must be a block. A block is defined as:

[<<label>>]
[DECLARE Declarations]
BEGIN
statements
END [label];

Each declaration and each statement within a block is terminated by a semicolon. A block that appears within another block must have a semicolon after END , as shown above; however the final END that concludes a function body does not require a semicolon

IF boolean-expression THEN
statements
END IF;
	IF boolean-expression
	THEN
	statements
	ELSE statements
	END IF;

WHILE boolean-expression LOOP statements END LOOP [label];	FOR name IN [REVERSE] expression..expression [BY expression] LOOP statements END LOOP [label]; FOR i IN 1..10 LOOP -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop END LOOP; FOR i IN REVERSE 10..1 LOOP -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop END LOOP; FOR i IN REVERSE 10..1 BY 2 LOOP -- i will take on the values 10,8,6,4,2 within the loop END LOOP;

Procedure

1. Write a PROCEDURE to display sum of digits of a three digit number

2. Write a procedure/ block to display prime numbers

```
Input : N = 20
Output : 2, 3, 5, 7, 11, 13, 17, 19
```

3. Write a procedure/block to display Fibonacci series upto 8th term (start with 0,1)
4. Create or use EMP(eid, Name, location, mid). Write a procedure using cursor to display list of managers(mid) with name;

Post Lab Questions:	1. Give advantages of PLSQL vs SQL 2. Explain data types of PgSQL/plsql of mysql

Q1.Write a PROCEDURE to display sum of digits of a three digit number

dbms2023/s9913@dbms2023

No limit

Query Query History

```
create function SumOfDigits(number INT)
returns int as $$
declare
    digit1 int;
    digit2 int;
    digit3 int;
    sum INT;
begin
    digit1 := @number/100;
    digit2 := (@number/10)%10;
    digit3 := @number%10;
    sum := digit1+digit2+digit3;
    return sum;
end;
$$ language plpgsql;

select SumOfDigits(123)
```

Data output Messages Notifications

	sumofdigits integer
1	6

Q2 Write a procedure/ block to display prime numbers

```
CREATE OR REPLACE FUNCTION generate_primes(limit_num INT)
RETURNS SETOF INT AS $$
DECLARE
        num INT;
        divisor INT;
        is_prime BOOLEAN;
BEGIN
        num := 2; -- Starting from 2, as it's the smallest prime number

        WHILE num <= limit_num LOOP
        is_prime := TRUE;

        -- Check if num is divisible by any number other than 1 and itself
        FOR divisor IN 2..ROUND(SQRT(num)) LOOP
        IF num % divisor = 0 THEN
        is_prime := FALSE;
        EXIT;
```

```
        END IF;
        END LOOP;

        IF is_prime THEN
        RETURN NEXT num;
        END IF;

        num := num + 1;
        END LOOP;

        RETURN;
END;
$$ LANGUAGE PLPGSQL;

select generate_primes(20)
```

	generate_primes integer
1	2
2	3
3	5
4	7
5	11
6	13
7	17
8	19

Q3 Write a procedure/block to display Fibonacci series upto 8th term (start with 0,1)

```
CREATE OR REPLACE FUNCTION fibonacci(limit_num INT)
RETURNS SETOF INT AS $$
DECLARE
   num1 int := 0;
   num2 int := 1;
        num3 INT;
   n int := 2;
BEGIN
   return next 0;
   return next 1;
   while n<limit_num LOOP
         num3 := num1 + num2;
         num1 := num2;
         num2 := num3;
         n := n+1;

         RETURN next num3;
```

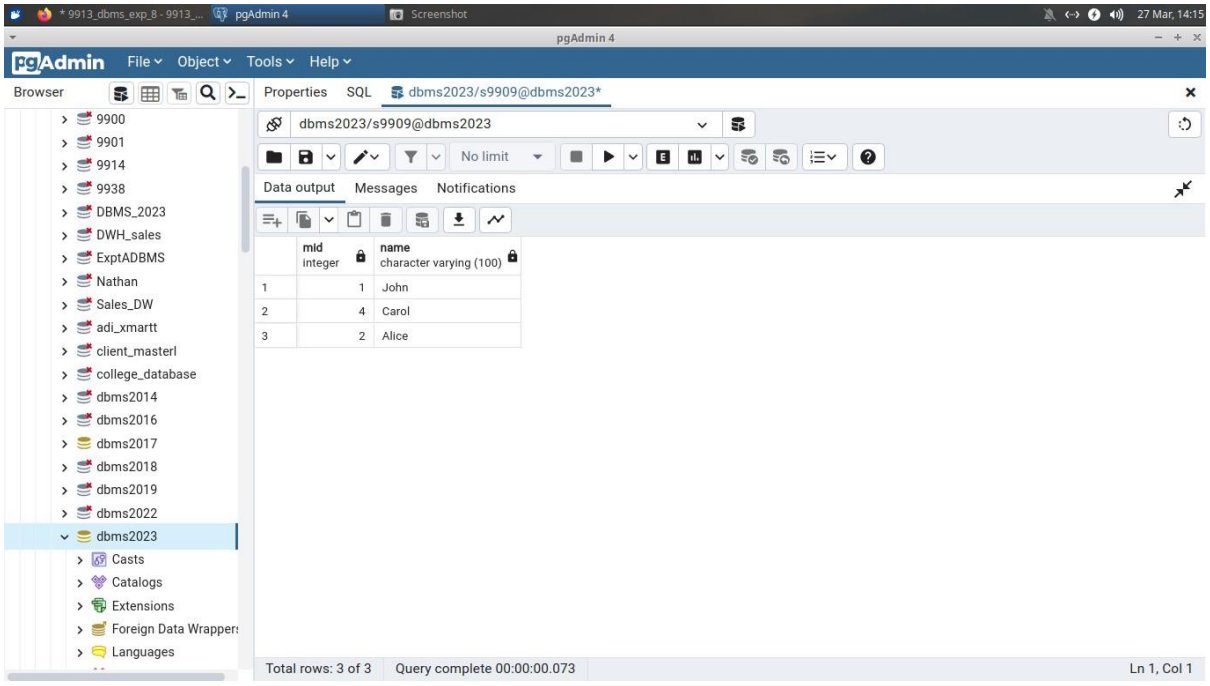
```
    end loop;
    return;
END;
$$ LANGUAGE PLPGSQL;

select fibonacci(8)
```

	fibonacci integer
1	0
2	1
3	1
4	2
5	3
6	5
7	8
8	13

Q4 Create or use EMP(eid, Name, location, mid). Write a procedure using cursor to display list of managers(mid) with name;

```
CREATE TABLE EMP (
eid SERIAL PRIMARY KEY,Name
VARCHAR(100),
location VARCHAR(100),mid INT,
FOREIGN KEY (mid) REFERENCES EMP(eid) -- Self-reference
);
-- Insert sample data
INSERT INTO EMP (Name, location, mid) VALUES ('John', 'New York', 1); INSERT INTO EMP
(Name, location, mid) VALUES ('Alice', 'Los Angeles', 2); INSERT INTO EMP (Name, location,
mid) VALUES ('Bob', 'Chicago', 1); INSERT INTO EMP (Name, location, mid) VALUES ('Carol',
'Houston', NULL);INSERT INTO EMP (Name, location, mid) VALUES ('David', 'Boston', 4);
-- Join EMP with itself based on eid=midSELECT
distinct e1.mid , e2.Name FROM EMP e1
join emp e2
on e1.mid = e2.eid
```



Postlab

Q1

PL/SQL offers several advantages over SQL:

- Procedural Capabilities: PL/SQL provides procedural constructs such as loops, conditional statements, exception handling, and subprograms like functions and procedures. This allows for more complex logic to be implemented directly within the database, reducing the need for round-trips between the application and the database server.
- Encapsulation and Modularity: PL/SQL allows for the encapsulation of SQL statements within blocks of code. This promotes modularity and code reusability, making it easier to maintain and update database logic.
- Performance Optimization: PL/SQL can improve performance by reducing the number of interactions between the application and the database. By executing multiple SQL statements within a single PL/SQL block, you can minimize network traffic and reduce overhead.
- Enhanced Error Handling: PL/SQL provides robust error handling mechanisms, including exception handling blocks, which allow for graceful handling of errors within the database. This improves the reliability and maintainability of database applications.

Q2 Explain data types of PgSQL/plsql of mysql

1. Numeric Data Types:

- PgSQL: Includes integer types like int, smallint, bigint, and floating-point types like real, double precision.
- PL/SQL: Offers similar numeric types such as INTEGER, SMALLINT, NUMBER, and FLOAT.
- MySQL: Provides numeric types like INT, SMALLINT, BIGINT, FLOAT, DOUBLE, etc.

2. Character Data Types:

- PgSQL: Offers character varying(n) (VARCHAR), character(n) (CHAR), text, etc.
- PL/SQL: Provides CHAR, VARCHAR2, CLOB for character data.
- MySQL: Supports CHAR, VARCHAR, TEXT, etc.

3. Date and Time Data Types:

- PgSQL: Includes timestamp, date, time, interval, etc.
- PL/SQL: Offers DATE, TIMESTAMP, INTERVAL for date and time handling.
- MySQL: Provides DATE, TIME, DATETIME, TIMESTAMP, YEAR, etc.