EXPERIMENT - VIII
PL/SQL AND SEQUENCE

August 18, 2019

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To study the basic PL/SQL and sequence queries.

PL/SQL

PL/SQL, Oracle's procedural extensions to SQL, is an advanced fourth-generation programming language (4GL). It offers modern features such as data encapsulation, overloading, collection types, exception handling, and information hiding. PL/SQL also offers seamless SQL access, tight integration with the Oracle server and tools, portability, and security.

Block Structure

PL/SQL is a block-structured language. That is, the basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or subproblem to be solved. Thus, PL/SQL supports the divide-and-conquer approach to problem solving called stepwise refinement.
A block (or sub-block) lets you group logically related declarations and statements. That way, you can place declarations close to where they are used. The declarations are local to the block and cease to exist when the block completes.
A PL/SQL block has three parts: a declarative part, an executable part, and an exception-handling part. (In PL/SQL, a warning or error condition is called an exception.) Only the executable part is required.
The order of the parts is logical. First comes the declarative part, in which items can be declared. Once declared, items can be manipulated in the executable part. Exceptions raised during execution can be dealt with in the exception-handling part.
You can nest sub-blocks in the executable and exception-handling parts of a PL/SQL block or subprogram but not in the declarative part. Also, you can define local subprograms in the declarative part of any block. However, you can call local subprograms only from the block in which they are defined.

SEQUENCES

The sequence generator generates sequential numbers. Sequence number generation is useful to generate unique primary keys for your data automatically, and to coordinate keys across multiple rows or tables.
Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as serialization. If you have such constructs in your applications, then you should replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of your application.

QUESTIONS

Write PL/SQL programs for the following:

1. To print the first 'n' prime numbers

SOURCE CODE:

```
postgres=# CREATE OR REPLACE FUNCTION PRIME(N INT)
postgres-# RETURNS SETOF INT AS
postgres-# $$
postgres$# DECLARE
postgres$# i INT;
postgres$# j INT;
postgres$# counter INT;
postgres$# BEGIN
postgres$# counter:=0;
postgres$# i:=2;
postgres$# LOOP
postgres$# j:=2;
postgres$# LOOP
postgres$# EXIT WHEN((i%j=0)OR(j=i));
postgres$# j:=j+1;
postgres$# END LOOP;
postgres$# IF(j=i) THEN
postgres$# RETURN NEXT i;
postgres$# counter:=counter+1;
postgres$# END IF;
postgres$# i:=i+1;
postgres$# EXIT WHEN counter=n;
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT:

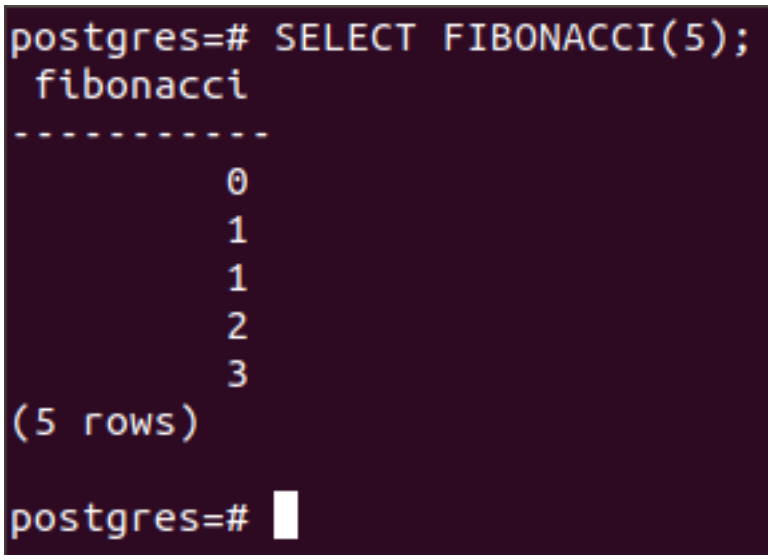
```
postgres=# SELECT PRIME(20);
 prime
-------
     2
     3
     5
     7
    11
    13
    17
    19
    23
    29
    31
    37
    41
    43
    47
    53
    59
    61
    67
    71
(20 rows)

postgres=#
```

2. Display the Fibonacci series upto 'n' terms

SOURCE CODE:

```
postgres=# CREATE OR REPLACE FUNCTION FIBONACCI(N INT)
postgres-# RETURNS SETOF INT AS
postgres-# $$
postgres$# DECLARE
postgres$# a INT;
postgres$# b INT;
postgres$# c INT;
postgres$# BEGIN
postgres$# a:=0;
postgres$# b:=1;
postgres$# FOR counter IN 1..n LOOP
postgres$# RETURN NEXT a;
postgres$# c:=a+b;
postgres$# a:=b;
postgres$# b:=c;
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT:


postgres=# SELECT FIBONACCI(5);
fibonacci
0
1
1
2
3
(5 rows)
postgres=#

3. Create a table named student_grade with the given attributes: roll, name ,mark1,mark2,mark3, grade. Read the roll, name and marks from the user. Calculate the grade of the student and insert a tuple into the table using PL/SQL.(Grade= 'PASS' if AVG >40, Grade='FAIL' otherwise)

SOURCE CODE:

```
postgres=# CREATE FUNCTION GRADE() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# REC RECORD;
postgres$# m1 int;
postgres$# m2 int;
postgres$# m3 int;
postgres$# avg numeric;
postgres$# BEGIN
postgres$# FOR REC IN SELECT * FROM STUDENT_GRADE
postgres$# LOOP
postgres$# m1:=REC.MARK1;
postgres$# m2:=REC.MARK2;
postgres$# m3:=REC.MARK3;
postgres$# avg=(m1+m2+m3)/3;
postgres$# IF(AVG>40) THEN
postgres$# UPDATE STUDENT_GRADE SET GRADE='PASS'
postgres$# WHERE ROLL=REC.ROLL;
postgres$# ELSE
postgres$# UPDATE STUDENT_GRADE SET GRADE='FAIL'
postgres$# WHERE ROLL=REC.ROLL;
postgres$# END IF;
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT:

```
postgres=# SELECT * FROM STUDENT_GRADE;
 roll | name | mark1 | mark2 | mark3 | grade
------+------+-------+-------+-------+-------
    1 | Anu  |    50 |    45 |    48 |
    2 | Manu |    50 |    50 |    50 |
    3 | Manu |    35 |    40 |    40 |
(3 rows)

postgres=# SELECT GRADE();
 grade
-------

(1 row)

postgres=# SELECT * FROM STUDENT_GRADE;
 roll | name | mark1 | mark2 | mark3 | grade
------+------+-------+-------+-------+-------
    1 | Anu  |    50 |    45 |    48 | PASS
    2 | Manu |    50 |    50 |    50 | PASS
    3 | Manu |    35 |    40 |    40 | FAIL
(3 rows)

postgres=#
```

4. Create table circle_area (rad,area). For radius 5, 10, 15, 20 & 25, find the area and insert the corresponding values into the table by using loop structure in PL/SQL.

SOURCE CODE:

```
postgres=# CREATE OR REPLACE FUNCTION FIND_AREA()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# REC RECORD;
postgres$# R INT;
postgres$# BEGIN
postgres$# FOR I IN 1..5 LOOP
postgres$# R:=5*I;
postgres$# INSERT INTO CIRCLE_AREA VALUES(R,3.14*R*R);
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT:

```
postgres=# SELECT * FROM CIRCLE_AREA;
 radius | area
--------+------
(0 rows)

postgres=# SELECT FIND_AREA();
 find_area
-----------

(1 row)

postgres=# SELECT * FROM CIRCLE_AREA;
 radius |  area
--------+---------
      5 |   78.50
     10 |  314.00
     15 |  706.50
     20 | 1256.00
     25 | 1962.50
(5 rows)

postgres=#
```

5. Use an array to store the names, marks of 10 students in a class. Using Loop structures in PL/SQL insert the ten tuples to a table named stud.

SOURCE CODE:

```
postgres=# CREATE OR REPLACE FUNCTION GETDATA()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# NAME TEXT[]=ARRAY['ARUN','AMAL','PETER','JOSE','ANNIE',
postgres$# 'MARY','JOSEPH','MARK','MIDHUN','KEVIN'];
postgres$# MARKS INT[]=ARRAY[25,76,43,45,67,57,97,56,89,8];
postgres$# BEGIN
postgres$# FOR i IN 1..10 LOOP
postgres$# INSERT INTO STUD VALUES(NAME[i],MARKS[i]);
postgres$# END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT:

```
postgres=# SELECT * FROM STUD;
 name | marks
------+-------
(0 rows)

postgres=# SELECT GETDATA();
 getdata
---------

(1 row)

postgres=# SELECT * FROM STUD;
  name   | marks
---------+-------
 ARUN    |    25
 AMAL    |    76
 PETER   |    43
 JOSE    |    45
 ANNIE   |    67
 MARY    |    57
 JOSEPH  |    97
 MARK    |    56
 MIDHUN  |    89
 KEVIN   |     8
(10 rows)

postgres=#
```

6. Create a sequence using PL/SQL. Use this sequence to generate the primary key values for a table named class_cse with attributes roll, name and phone. Insert some tuples using PL/SQL programming.

SOURCE CODE:

```
postgres=# CREATE TABLE CLASS_CSE(
postgres(# ROLL SERIAL,
postgres(# NAME TEXT,
postgres(# PHONE TEXT);
CREATE TABLE
postgres=#
```

OUTPUT:

```
postgres=# INSERT INTO CLASS_CSE(NAME,PHONE)
postgres-# VALUES('Arun','0482-239091');
INSERT 0 1
postgres=# INSERT INTO CLASS_CSE(NAME,PHONE)
VALUES('Amal','0484-234562');
INSERT 0 1
postgres=# INSERT INTO CLASS_CSE(NAME,PHONE)
VALUES('Peter','0485-11234');
INSERT 0 1
postgres=# INSERT INTO CLASS_CSE(NAME,PHONE)
VALUES('Jose','0489-43617');
INSERT 0 1
postgres=# INSERT INTO CLASS_CSE(NAME,PHONE)
VALUES('Annie','0481-23145');
INSERT 0 1
postgres=# SELECT * FROM CLASS_CSE;
 roll | name  |    phone
------+-------+-------------
    1 | Arun  | 0482-239091
    2 | Amal  | 0484-234562
    3 | Peter | 0485-11234
    4 | Jose  | 0489-43617
    5 | Annie | 0481-23145
(5 rows)

postgres=#
```

RESULT

The PL/SQL program was executed successfully and the output was obtained.