COURSE NAME: DBMS
COURSE CODE:23AD2102A

TOPIC:

PL/SQL FOR PROCEDURAL PROGRAMMING

Session - 14

AIM OF THE SESSION

To familiarize students with the advance and complex Subqueries in PostgreSQL.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Discuss the subqueries.
2. Various guidelines and types of subqueries.

LEARNING OUTCOMES

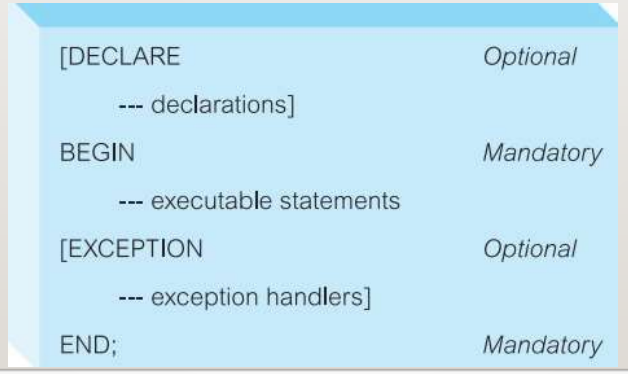
At the end of this session, you should be able to understand the basic concepts of Subqueries and learn how to write complex subqueries with PostgreSQL commands.

PROCEDURES IN PL/SQL

- PL/SQL has concepts similar to modern programming languages, such as variable and constant declarations, control structures, exception handling, and modularization.
- PL/SQL is a block-structured language: blocks can be entirely separate or nested within one another. The basic units that constitute a PL/SQL program are procedures, functions, and anonymous (unnamed) blocks.

PL/SQL BLOCK

- PL/SQL block has up to three parts:
- • an optional declaration part, in which variables, constants, cursors, and exceptions are defined and possibly initialized.
- • a mandatory executable part, in which the variables are manipulated.
- • an optional exception part, to handle any exceptions raised during execution.



DECLARATIONS

```
vStaffNo VARCHAR2(5);
vRent NUMBER(6, 2) NOT NULL := 600;
MAX_PROPERTIES CONSTANT NUMBER := 100;
```

%TYPE and %ROWTYPE attributes:

```
vStaffNo Staff.staffNo%TYPE;
vStaffNo1 vStaffNo%TYPE;
```

Declaring variable of the type of a column of specified table OR a type of other variable.

```
vStaffRec Staff%ROWTYPE;
```

declaring a variable to be of the same type as a entire row of a table or view.

ASSIGNMENTS

```
vStaffNo := 'SG14';
vRent := 500;
SELECT COUNT(*) INTO x FROM PropertyForRent WHERE staffNo = vStaffNo;
```

In the last statement, the variable x is set to the result of the SELECT statement.

CONTROL STATEMENTS

- PL/SQL supports the usual conditional, iterative, and sequential flow-of-control mechanisms.
- **Conditional IF statement**

```
IF (position = 'Manager') THEN
    salary := salary*1.05;
ELSE
    salary := salary*1.03;
END IF;
```

Conditional CASE statement

```
UPDATE Staff
SET salary = CASE
             WHEN position = 'Manager'
               THEN salary * 1.05
             ELSE
               THEN salary * 1.02
             END;
```

ITERATION STATEMENTS

(LOOP)

```
x:=1;
myLoop:
LOOP
   x := x+1;
   IF (x > 3) THEN
         EXIT myLoop;          --- exit loop immediately
END LOOP myLoop;
--- control resumes here
y := 2;
```

WHILE and REPEAT

PL/SQL

```
WHILE (condition) LOOP
        <SQL statement list>
END LOOP [labelName];
```

SQL

```
WHILE (condition) DO
        <SQL statement list>
END WHILE [labelName];
REPEAT
        <SQL statement list>
UNTIL (condition)
END REPEAT [labelName];
```

PL/SQL

```
FOR indexVariable
  IN lowerBound .. upperBound LOOP
  <SQL statement list>
END LOOP [labelName];
```

SQL

```
FOR indexVariable
  AS querySpecification DO
  <SQL statement list>
END FOR [labelName];
```

CURSORS IN PL/SQL

- SELECT statement can be used if the query returns one and only one row.
- To handle a query that can return an arbitrary number of rows (that is, zero, one, or more rows) PL/SQL uses cursors to allow the rows of a query result to be accessed one at a time.
- In effect, the cursor acts as a pointer to a particular row of the query result.
- The cursor can be advanced by 1 to access the next row.
- A cursor must be declared and opened before it can be used, and it must be closed to deactivate it after it is no longer required.
- Once the cursor has been opened, the rows of the query result can be retrieved one at a time using a FETCH statement, as opposed to a SELECT statement.

EXAMPLE OF CURSOR

```
DECLARE
        vPropertyNo         PropertyForRent.propertyNo%TYPE;
        vStreet             PropertyForRent.street%TYPE;
        vCity               PropertyForRent.city%TYPE;
        vPostcode           PropertyForRent.postcode%TYPE;
        CURSOR propertyCursor IS
                SELECT propertyNo, street, city, postcode
                FROM PropertyForRent
                WHERE staffNo = 'SG14'
                ORDER by propertyNo;
BEGIN
-- Open the cursor to start of selection, then loop to fetch each row of the result table
        OPEN propertyCursor;
        LOOP

-- Fetch next row of the result table
        FETCH propertyCursor
                INTO vPropertyNo, vStreet, vCity, vPostcode;
        EXIT WHEN  propertyCursor%NOTFOUND;

-- Display data
        dbms_output.put_line('Property number: ' || vPropertyNo);
        dbms_output.put_line('Street:          ' || vStreet);
        dbms_output.put_line('City:            ' || vCity);
        IF postcode IS NOT NULL THEN
                dbms_output.put_line('Post Code:          ' || vPostcode);
        ELSE
                dbms_output.put_line('Post Code:          NULL');
        END IF;
    END LOOP;
    IF propertyCursor%ISOPEN THEN CLOSE propertyCursor END IF;

-- Error condition - print out error
EXCEPTION
  WHEN OTHERS THEN
        dbms_output.put_line('Error detected');
        IF propertyCursor%ISOPEN THEN CLOSE propertyCursor; END IF;
END;
```

PARAMETERIZED CURSORS

- Passing parameters to cursors PL/SQL allows cursors to be parameterized, so that the same cursor definition can be reused with different criteria.

```
CURSOR propertyCursor (vStaffNo VARCHAR2) IS
        SELECT propertyNo, street, city, postcode
        FROM PropertyForRent
        WHERE staffNo = vStaffNo
        ORDER BY propertyNo;
```

and we could open the cursor using the following example statements:

```
vStaffNo1 PropertyForRent.staffNo%TYPE := 'SG14';
OPEN propertyCursor('SG14');
OPEN propertyCursor('SA9');
OPEN propertyCursor(vStaffNo1);
```

SUBPROGRAMS, STORED PROCEDURES, FUNCTIONS, AND PACKAGES

- Subprograms are named PL/SQL blocks that can take parameters and be invoked.
- PL/SQL has two types of subprogram called (stored) procedures and functions.
- Procedures and functions can take a set of parameters given to them by the calling program and perform a set of actions.
- Both can modify and return data passed to them as a parameter.
- The difference between a procedure and a function is that a function will always return a single value to the caller, whereas a procedure does not.
- Usually, procedures are used unless only one return value is needed.

STORED PROCEDURES AND FUNCTIONS

- Procedures and functions are very similar to those found in most high-level programming languages and have the same advantages:
- This reduces duplication of effort and improves software modularity and extensibility.
- promote reusability and maintainability
- aid abstraction
- In databases, many applications can access the procedure stored on the server. Server acts as a single point of change.
- Stored procedures and functions allow business logic to be stored in databases which can be invoked from SQL statements.

GENERAL FORM FOR CREATING AND CALLING PROCEDURES AND FUNCTIONS

```
CREATE PROCEDURE <procedure name> ( <parameters> )
<local declarations>
<procedure body> ;
```

```
CREATE FUNCTION <function name> ( <parameters> )
RETURNS <return type>
<local declarations>
<function body> ;
```

```
CALL <procedure or function name> ( <argument list> ) ;
```

Read each Create as Create or Replace

PARAMETER TYPES

- A parameter has a specified name and data type but can also be designated as:
- • IN – parameter is used as an input value only.
- • OUT – parameter is used as an output value only.
- • IN OUT – parameter is used as both an input and an output value.

EXAMPLE OF CREATE FUNCTION AND FUNCTION CALL

```
create function instructor_of (dept_name varchar(20))
        returns table (
                ID varchar (5),
                name varchar (20),
                dept_name varchar (20),
                salary numeric (8,2))
return table
        (select ID, name, dept_name, salary
        from instructor
        where instructor.dept_name = instructor_of.dept_name);
```

```
select *
from table(instructor_of ('Finance'));
```

This query returns all instructors of the 'Finance' department.

EXAMPLE OF CREATE PROCEDURE AND CALL PROCEDURE

```
create procedure dept_count_proc(in dept_name varchar(20),
                                 out d_count integer)
    begin
        select count(*) into d_count
        from instructor
        where instructor.dept_name= dept_count_proc.dept_name
    end
```

```
declare d_count integer;
call dept_count_proc('Physics', d_count);
```

query set the d_count variable of the dept_count_proc procedure with the number of instructors of Physics department.

The procedure could also be executed in SQL*Plus as:

```
SQL> SET SERVEROUTPUT ON;
SQL> EXECUTE PropertiesForStaff('SG14');
```

PACKAGES IN PL/SQL

- Packages (PL/SQL) A package is a collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit.
- A package has two parts: a specification and a body.
- A package's specification declares all public constructs of the package, and the body defines all constructs (public and private) of the package, and so implements the specification.
- Oracle database performs the following steps when a procedure or package is created:
- • It compiles the procedure or package.
- • It stores the compiled code in memory.
- • It stores the procedure or package in the database.

EXAMPLES

- CREATE OR REPLACE PACKAGE StaffPropertiesPackage AS procedure PropertiesForStaff(vStaffNo VARCHAR2); END StaffPropertiesPackage;
- and we could create the package body (that is, the implementation of the package) as:
- CREATE OR REPLACE PACKAGE BODY StaffPropertiesPackage AS . . . END StaffPropertiesPackage;
- To reference the items declared within a package specification, we use the dot notation. For example, we could call the PropertiesForStaff procedure as follows: StaffPropertiesPackage.PropertiesForStaff('SG14');
-

SUMMARY

An aggregate function in SQL performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function

SELF-ASSESSMENT QUESTIONS

1. Which of the following is true about sub-queries?

a) They execute after the main query executes.
b) They execute in parallel to the main query.
c) The user can execute the main query and then, if wanted, execute the sub-query.
d) They execute before the main query executes.

2. Which of the following clause is mandatorily used in a sub-query?

(a) SELECT
(b) WHERE
(c) ORDER BY
(d) GROUP BY

SELF-ASSESSMENT QUESTIONS

3. Which of the following multi-row operators can be used with a sub-query?

(a) IN
(b) ANY
(c) ALL
(d) ALL OF THE ABOVE

4. Which of the following is true about the result of a sub-query?

a) The result of a sub-query is generally ignored when executed.
b) The result of a sub-query doesn't give a result, it is just helpful in speeding up the main query execution.
c) The result of a sub-query is used by the main query.
d) The result of a sub-query is always NULL.

TERMINAL QUESTIONS

1. Describe various types of SQL complex subqueries.

2. List out the guidelines for creating the SQL subqueries.

3. Analyze the use of ALL,IN, or ANY operator while using subqueries in PostgreSQL.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. Database System Concepts, Sixth Edition, Abraham Silberschatz, Yale University Henry, F. Korth Lehigh University, S. Sudarshan Indian Institute of Technology, Bombay.

2. An Introduction to Database Systems by Bipin C. Desai

3. Fundamentals of Database Systems, 7th Edition, RamezElmasri, University of Texas at Arlington, Shamkant B. Navathe, University of Texasat Arlington.

Sites and Web links:

1. https://www.geeksforgeeks.org/postgresql-create-table/

2. https://www.tutorialsteacher.com/postgresql

THANK YOU

Team – DBMS