COURSE NAME: DBMS
COURSE CODE:23AD2102R

TOPIC:

PL/SQL FOR PROCEDURAL PROGRAMMING

Session - 17

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 BASIC SYNTAX

```
DECLARE
      <declarations section>
BEGIN
      <executable command(s)>
EXCEPTION
      <exception handling>
END;
```

PL/SQL BLOCK DESCRIPTION

- **Declarations**
  - This section starts with the keyword DECLARE.
  - It is an optional section and defines all variables, cursors, etc.
- **Executable Commands**
  - It's enclosed between keywords BEGIN and END.
  - It's a mandatory section.
  - It consists of executable PL/SQL statements of the program.
- **Exception Handling**
  - This section starts with the keyword EXCEPTION.
  - This section is also an optional section.
  - It contains exceptions that handle errors in the program.

Advantages of PL/SQL

Procedural Capabilities – Unlike SQL, which is declarative, PL/SQL supports procedural constructs like loops, conditionals, and exception handling, making complex logic easier to implement.

Improved Performance – PL/SQL reduces the number of database calls by allowing multiple SQL operations to be executed as a single block, reducing network overhead.

Code Reusability – PL/SQL supports stored procedures, functions, packages, and triggers, promoting modular programming and code reuse.

Better Security – Permissions and privileges can be assigned to stored procedures and packages, ensuring that only authorized users can execute certain operations.

Exception Handling – PL/SQL provides robust error-handling mechanisms, allowing developers to manage runtime errors gracefully.

Concurrency Control – With built-in locking mechanisms, PL/SQL helps in handling concurrent transactions effectively.

Portability – PL/SQL code can be used across different platforms where Oracle Database runs.

Disadvantages of PL/SQL

Limited Portability – While PL/SQL is powerful within Oracle databases, it is not directly compatible with other databases like MySQL or SQL Server.

Complex Debugging – Debugging PL/SQL code can be challenging, especially when dealing with stored procedures and triggers, as they run inside the database.

Steeper Learning Curve – PL/SQL requires knowledge of both SQL and procedural programming, making it harder for beginners to master.

Performance Overhead – While PL/SQL reduces network traffic, executing large amounts of procedural code inside the database may sometimes be slower compared to optimized SQL queries.

Vendor Lock-in – Since PL/SQL is Oracle-specific, organizations using it are dependent on Oracle technologies, which can be costly and limit migration options.

SIMPLE PL/SQL PROGRAM

Print natural numbers from 1 to 5.

```
declare
    i number;
begin
    i:=1;
    loop
    dbms_output.put_line(i);
    i:=i+1;
    exit when i > 5;
    end loop;
end;
```

PROCEDURES

- It's a named block of statement.
- It may or may not return a value.

SYNTAX:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, …])]
{IS | AS}
BEGIN
      <procedure body>
END procedure_name;
```

SIMPLE PROGRAM USING PROCEDURE

```
create or replace procedure topperStudent
as
name students.s_name%type;
begin
select name from student where marks=(select max(marks) from student)
dbms_output.put_line(name);
end;
```

- To execute, there are two ways to do it:
  - 1) exec topperStudent;
  - 2) begin topperStudent end;

CURSOR

- A cursor is a temporary area created in the main memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.
- There are two types of cursors in PL/SQL:
  - Implicit
  - Explicit

IMPLICIT CURSOR

- Implicit cursors get created when you execute DMA queries like Select, insert, delete, update.
- Oracle gives some useful attributes on this implicit cursors to help us check the status of DML operations

Attribute	Usage
%FOUND	If DML statement affects at least one row returns TRUE else returns FALSE
%NOTFOUND	If DML statement affects at least one row returns FALSE else returns TRUE
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT

EXAMPLE OF IMPLICIT CURSOR

```
create or replace procedure updateFees(newFee int)
as
var_rows number;
begin
update student set fees=newFee;
if SQL%FOUND then
   var_rows :=SQL%ROWCOUNT;
   dbms_output.put_line('The fees of '|| var_rows || ' students was updated');
else
   dbms_output.put_line('Some issue in updating');
end if;
end;
```

EXPLICIT CURSOR

- They must be created when you are executing a SELECT statement that returns more than one row in a PL/SQL procedure or a function.
- Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.
- Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

EXPLICIT CURSOR : EXAMPLE 1

- Show the average fees paid by students of each department.

```
declare cursor c1
is
select deptName as Department,avg(fees) as Average_Fees from student natural join
    department group by deptName;
rec1 c1%rowtype;
begin
for rec1 in c1 loop
dbms_output.put_line(rec1.Department ||' '||rec1.Average_Fees);
end loop;
end;
```

EXPLICIT CURSOR : EXAMPLE 2

Show the department wise student details.

```
declare cursor c1
is
select distinct deptName from department;
cursor c2(dept varchar)
is
select name,marks from student natural join department where deptName=dept ;
rec1 c1%rowtype;
rec2 c2%rowtype;
begin
for rec1 in c1 loop
dbms_output.put_line(rec1.deptName);
for rec2 in c2(rec1.deptname) loop
dbms_output.put_line(rec2.name||' '||rec2.marks);
end loop;
dbms_output.put_line('');
end loop;
end;
```

PROCEDURE WITH CURSOR

- Statement:Write a procedure which will display details of all students from a given department:

```
create or replace procedure listStudents(dept varchar)
is
cursor c1 is
select rollNo,name,marks from student natural join department where deptName=dept;
rec1 c1%rowtype;
begin
for rec1 in c1 loop
dbms_output.put_line(rec1.rollNo||' '||rec1.name||' '||rec1.marks);
end loop;
end;
```

To Execute:

```
begin
listStudents('production');
end;
```

PL/SQL FUNCTIONS

- A PL/SQL function is same as a procedure except that it always returns a value.
- General Syntax:
  - CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])] RETURN return_datatype {IS | AS}

    BEGIN

    < function_body >

    END [function_name];

FUNCTIONS: EXAMPLE 1

- Write a function which will return the total fees collected for a given department.

```
create or replace function totalFees(dept varchar)
return int
is
total int;
begin
select sum(fees) into total from student natural join department where deptName=dept;
return total;
end;
```

To execute: select totalFees('production') from dual;

FUNCTIONS: EXAMPLE 2

- Get the topper student from a given department

```
create or replace function getTopper(dept varchar)
return varchar
is
topper varchar(50);
begin
select name into topper from student natural join department where deptName=dept and
    marks=(select max(marks) from student natural join department where
    deptName=dept) ;
return topper;
end;
```

- To execute: select getTopper('Civil') from dual;

TRIGGERS

- Triggers are stored routines, which are automatically executed when some events occur.
- Triggers are written to be executed in response to any of the following events:
    - DML - DELETE, INSERT, or UPDATE.
    - DDL - CREATE, ALTER, or DROP .
    - A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

TRIGGER: GENERAL SYNTAX

CREATE [OR REPLACE] TRIGGER *trigger_name* BEFORE/AFTER
INSERT/UPDATE/DELETE

ON *table_name* [FOR EACH ROW]

DECLARE

-- variable declarations

BEGIN

-- trigger code

EXCEPTION WHEN ...

-- exception handling

END;

TRIGGER : BEFORE INSERT

- Fired before the INSERT operation is executed.
- Check for email availability before inserting information of new student.

```
CREATE OR REPLACE TRIGGER checkEmail
BEFORE INSERT ON student FOR EACH ROW
declare
   rowcount int;
begin
    SELECT COUNT(*) into rowcount FROM student WHERE email = :NEW.email;
    IF rowcount<>0 THEN
       raise_application_error(-20001,'Email Already Registered');
    END IF;
END;
```

TRIGGER: AFTER INSERT

```
CREATE OR REPLACE TRIGGER cancelAdmit
AFTER INSERT ON student
REFERENCING NEW AS n
FOR EACH ROW
declare
   rowcount int;
begin
    if :n.fees <10000 then
  dbms_output.put_line('Admission cancelled due to less donation');
    end if;
END;
```

TRIGGER BEFORE UPDATE

- Give Notification to Admin of Email change.

```
CREATE OR REPLACE TRIGGER checkUpdatedEmail
BEFORE UPDATE ON student
REFERENCING NEW AS n
FOR EACH ROW
declare
   rowcount int;
begin
    dbms_output.put_line('The email has been changed to: ' || :n.email);
END;
```

TRIGGER AFTER DELETE

- Remove all students of the department, once the department is deleted.

```
CREATE OR REPLACE TRIGGER cleanStudents
AFTER DELETE ON department
FOR EACH ROW
declare
   dept int;
begin
   dept := :OLD.deptNo;
   delete from student where deptNo=dept;
END;
```

PACKAGE

- A **package** is a schema object that groups logically related PL/SQL types, items, and subprograms.
- A package will have two mandatory parts:
  - Package specification
    - It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package
  - Package body or definition
    - The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from code outside the package.

SYNTAX FOR PACKAGES

- Package Specification

```
CREATE PACKAGE cust_sal
AS
PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;
```

Execute this: exec cust_sal;

- Package body

```
CREATE OR REPLACE PACKAGE BODY cust_sal
AS
PROCEDURE find_sal(c_id customers.id%TYPE)
IS
c_sal customers.salary%TYPE;
BEGIN
SELECT salary INTO c_sal FROM customers WHERE id = c_id;
dbms_output.put_line('Salary: '|| c_sal);
 END find_sal;
 END cust_sal;
```

To Execute: exec cust_sal.find_sal(4);

EXCEPTIONS

- An error condition during a program execution is called an exception in PL/SQL.
- PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program
- There are two types of exceptions:
  - System-defined exceptions
  - User-defined exceptions

GENERAL SYNTAX

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

WHEN exception1 THEN

exception1-handling-statements

WHEN exception2 THEN

exception2-handling-statements

WHEN exception3 THEN

exception3-handling-statements .....

WHEN others THEN

exception3-handling-statements

END;

SYSTEM DEFINED EXCEPTION

```
DECLARE
dept department.deptNo%type:=10;
name department.deptName%type;
BEGIN
SELECT deptName into name FROM department where deptNo=dept;
DBMS_OUTPUT.PUT_LINE ('Name: '|| name);
EXCEPTION WHEN no_data_found THEN
dbms_output.put_line('No such department!');
WHEN others THEN
dbms_output.put_line('Error!');
END;
```

Here in this example, we have used system defined exception:

no_data_found

USER DEFINED EXCEPTION

```
DECLARE
   dept department.deptNo%type :=-23;
   name department.deptName%type;
   ex_invalid_deptNo  EXCEPTION;
BEGIN
   IF dept <= 0 THEN
      RAISE ex_invalid_deptNo;
   ELSE
      SELECT  deptName into name
      FROM department
      WHERE deptNo = dept;
      DBMS_OUTPUT.PUT_LINE (‘Department: '||  name);
   END IF;
EXCEPTION
   WHEN ex_invalid_deptNo THEN
      dbms_output.put_line('Department number must be greater than zero!');
   WHEN no_data_found THEN
      dbms_output.put_line('No such department!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
```

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