M.Sc. (Computer Science)- Second Semester Subject Name- "Practical Based on RDBMS(PL/SQL)" PL/SQL Cursor ad Package

By- Prof. Dileep Kumar Sahu

Assistant Professor

Department of Computer Application

Govt. Vishwanath Yadav Tamaskar Post Graduate Autonomous College, Durg (C.G.)

Email ID: dileepksahu20@gmail.com

Contents

- PL/SQL CURSOR
- PL/SQL PACKAGE

Objective

- We will learn how to create, compile, and execute a PL/SQL Cursor and package in Database.

PL/SQL Cursor

- When an SQL statement is processed, Oracle creates a memory area known as context area.
- A cursor is a pointer to this context area.
- It contains all information needed for processing the statement.
- In PL/SQL, the context area is controlled by Cursor.
- A cursor contains information on a select statement and the rows of data accessed by it.
- A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time.
- There are two types of cursors:

1. Implicit Cursors
2. Explicit Cursors

PL/SQL Cursor: PL/SQL Implicit Cursors

- The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.
- These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.
- Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations.
- Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

For example:

- When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor attributes tell whether any rows are affected and how many have been affected.
- If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

PL/SQL Cursor : Implicit Cursor

The following table specifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Cursor : Implicit Cursor - Example

- Create customers table and have records:
- A Program to update the table and increase salary of each customer by 5000.
- **Create procedure:**

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' Customers updated ');
  END IF;
END;
/
```

Here: SQL%ROWCOUNT attribute is used to determine the number of rows affected.

Customer			
Id	**Name**	**Department**	**Salary**
1	Ramesh	web developer	45000
2	Sohan	program developer	55000
3	Mohan	web designer	45000

Output: 3 Customers updated

After Execution of cursor the table will be updated as :

Customer			
Id	**Name**	**Department**	**Salary**
1	Ramesh	web developer	55000
2	Sohan	program developer	65000
3	Mohan	web designer	55000

PL/SQL Cursor: Explicit Cursors

- The Explicit cursors are defined to get more control over the context area.
- These cursors should be defined in the declaration section of the PL/SQL block.
- It is created on a SELECT statement which returns more than one row.

1. Declare the cursor:
   - to initialize in the memory.
   - It defines the cursor with a name and the associated SELECT statement.

- **Syntax:**

```
CURSOR name IS
  SELECT statement;
```

Steps to create Explicit Cursors

1. Declare the cursor:
    - to initialize in the memory.
    - It defines the cursor with a name and the associated SELECT statement.

- **Syntax:**

```
CURSOR name IS
  SELECT statement;
```

2. Open the cursor

- It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

Syntax for cursor open:

```
OPEN cursor_name;
```

Steps to create Explicit Cursors

3. Fetch the cursor:

- It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

Syntax for cursor fetch:

```
FETCH cursor_name INTO variable_list;
```

4. Close the cursor:

- It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

```
Close cursor_name;
```

PL/SQL Cursor: Explicit Cursors - Example

- Explicit cursors are defined by programmers to gain more control over the context area.
- It is defined in the declaration section of the PL/SQL block.
- It is created on a SELECT statement which returns more than one row.

PL/SQL Cursor : Explicit Cursor - Example

- **PL/SQL Program** to retrieve the customer name and address.

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Customer			
Id	**Name**	**Address**	**Salary**
1	Ramesh	Durg	45000
2	Sohan	Bhilai	55000
3	Mohan	Raipur	45000

Output:

```
1    Ramesh    Durg
2    Sohan     Bhilai
3    Mohan     Raipur

PL/SQL procedure successfully completed.
```

PL/SQL Package

- Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.
- A package will have two mandatory parts –
  1. Package specification
  2. Package body or definition

Package Specification

- The specification is the interface to the package.
- It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package.
- It contains all information about the content of the package, but excludes the code for the subprograms.
- All objects placed in the specification are called **public** objects.
- Any subprogram not in the package specification but coded in the package body is called a **private** object.

Creating PL/SQL Package

- The following code shows a package specification having a single procedure.
- We can have many global variables defined and multiple procedures or functions inside a package.

```
CREATE PACKAGE customer_sal AS
        PROCEDURE find_sal(c_id customer.id%type);
 END customer_sal
```

- Output:

```
Package created.
```

PL/SQL Package Body

- The **CREATE PACKAGE BODY** Statement is used for creating the package body. The following code shows the package body declaration for the ***customer_sal*** package created above.

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

- Output:

```
Package body created.
```

Using PL/SQL Package Elements

- The package elements (variables, procedures or functions) are accessed with the following syntax:

```
package_name.element_name;
```

- The following program uses the ***find_sal*** method of the ***customer_sal*** package –

```
DECLARE
    code customer.id%type := &cc_id;
BEGIN
    customer_sal.find_sal(code);
END;
/
```

- When the above code is executed at the SQL prompt, it prompts to enter the customer ID and when you enter an ID, it displays the corresponding salary as follows –

```
Enter value for cc_id: 1
Salary: 3000
PL/SQL procedure successfully completed.
```

PL/SQL Package: Example

- We will use the CUSTOMERS table stored in our database with the following records –
- Select * from customer;

Customer				
Id	**Name**	**Age**	**Address**	**Salary**
1	Ramesh	22	Durg	45000
2	Sohan	21	Bhilai	55000
3	Mohan	25	Raipur	45000

PL/SQL Package: Example

```
-----Creating Package
CREATE OR REPLACE PACKAGE cust_package AS
-- Add a customer
PROCEDURE addCustomer(cid customer.id%type,
cname customer.name%type,
cage customer.age%type,
caddr customer.address%type,
csal customer.salary%type);
-- Removes a customer
PROCEDURE delCustomer(cid customer.id%TYPE);
--Lists all customer
  PROCEDURE listCustomer;
END cust_package;
```

Output:

```
package created
```

PL/SQL Package: Example

```
-----Creating Package Body
CREATE OR REPLACE PACKAGE BODY cust_package AS
PROCEDURE addCustomer(cid customer.id%type,
cname customer.No.ame%type,
cage customer.age%type,
caddr customer.address%type,
csal customer.salary%type)
  IS
   BEGIN
         INSERT INTO customer (id,name,age,address,salary)
               VALUES(cid, cname, cage, caddr, csal);
 END addCustomer;
-------------------------------------------------------------------
 PROCEDURE delCustomer(cid customer.id%type)
IS
 BEGIN
      DELETE FROM customer WHERE id = cid;
  END delCustomer;
-------------------------------------------------------------------
PROCEDURE listCustomer IS
   CURSOR c_customer is
     SELECT name FROM customer;
       TYPE c_list is TABLE OF customer.Name%type;
       name_list c_list := c_list();
       counter integer :=0;
    BEGIN
      FOR n IN c_customer LOOP
           counter := counter +1;
            name_list.extend;
            name_list(counter) := n.name;
           dbms_output.put_line('Customer(' ||counter||
                    ')'||name_list(counter));
       END LOOP;
    END listCustomer;
END cust_package;
/
```

Output:

```
package body created
```

PL/SQL Package: Example

```
DECLARE code customer.id%type:= 2;
 BEGIN cust_package.addcustomer(4, 'Reyansh', 25, ‘Durg', 3500);
   cust_package.addcustomer(5, ‘Rishi', 32, ‘Raipur', 7500);
   cust_package.listcustomer;
   cust_package.delcustomer(code);
    cust_package.listcustomer;
END;
 /
```

PL/SQL Package: Example

Output:

```
Customer (1):       Ramesh
Customer (2):       Sohan
Customer (3):       Mohan
Customer (4):       Reyansh
Customer (5):       Rishi
Customer (1):       Ramesh
Customer (3):       Mohan
Customer (4):       Reyansh
Customer (5):       Rishi

PL/SQL Procedure successfully completed
```

Thank You