# 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:

* Implicit Cursors
* Explicit Cursors

## 1) 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.

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 Implicit Cursor Example

**Create customers table and have records:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

**Create procedure:**

1. **DECLARE**
2. total\_rows number(2);
3. **BEGIN**
4. **UPDATE** customers
5. **SET** salary = salary + 5000;
6. IF sql%notfound**THEN**
7. dbms\_output.put\_line('no customers updated');
8. ELSIF sql%found**THEN**
9. total\_rows := sql%rowcount;
10. dbms\_output.put\_line( total\_rows || ' customers updated ');
11. **END** IF;
12. **END**;
13. /

Output:

6 customers updated

PL/SQL procedure successfully completed.

Now, if you check the records in customer table, you will find that the rows are updated.

1. **select** \* **from** customers;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 25000 |
| 2 | Suresh | 22 | Kanpur | 27000 |
| 3 | Mahesh | 24 | Ghaziabad | 29000 |
| 4 | Chandan | 25 | Noida | 31000 |
| 5 | Alex | 21 | Paris | 33000 |
| 6 | Sunita | 20 | Delhi | 35000 |

## 2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain 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.**

Following is the syntax to create an explicit cursor:

## Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

1. **CURSOR** cursor\_name **IS** select\_statement;;

## Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

## 1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

**Syntax for explicit cursor decleration**

1. **CURSORnameIS**
2. **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:**

1. **OPEN**cursor\_name;

## 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:**

1. **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:**

1. **Close**cursor\_name;

## PL/SQL Explicit Cursor 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.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

**Create customers table and have records:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

**Create procedure:**

Execute the following program to retrieve the customer name and address.

1. **DECLARE**
2. c\_id customers.id%type;
3. c\_name customers.**name**%type;
4. c\_addrcustomers.address%type;
5. **CURSOR** c\_customers**is**
6. **SELECT** id, **name**, address **FROM** customers;
7. **BEGIN**
8. **OPEN**c\_customers;
9. LOOP
10. **FETCH**c\_customers**into**c\_id, c\_name, c\_addr;
11. EXIT **WHEN**c\_customers%notfound;
12. dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);
13. **END** LOOP;
14. **CLOSE**c\_customers;
15. **END**;
16. /

Output:

1 Ramesh Allahabad

2 Suresh Kanpur

3 Mahesh Ghaziabad

4 Chandan Noida

5 Alex Paris

6 Sunita Delhi

PL/SQL procedure successfully completed.

The %ROWTYPE attribute in PL/SQL allows you to declare a variable as a record that corresponds to a specific table or cursor. This means that the variable will have the same structure and data types as the table or cursor it is based on. This is useful when you want to manipulate data in a table or cursor without having to explicitly declare the individual fields and their data types.For example, if you have a table called "employees" with fields "id", "name", and "salary", you could declare a variable called "emp\_rec" as follows:[code]declare

emp\_recemployees%ROWTYPE;[/code]This would create a variable called "emp\_rec" that has the same structure as the "employees" table, with fields "id", "name", and "salary", and the corresponding data types. You can then use this variable to manipulate data in the "employees" table, such as adding, updating, or deleting records.

SQL> create table emp\_info(emp\_no number(10), emp\_name varchar2(20), emp\_dept number(10), emp\_salary number(20));

Table created.

SQL>descemp\_info;

Name Null? Type

----------------------------------------- -------- ----------------------------

EMP\_NO NUMBER(10)

EMP\_NAME VARCHAR2(20)

EMP\_DEPT NUMBER(10)

EMP\_SALARY NUMBER(20)

SQL>

SQL>

SQL>

SQL>

SQL>

SQL>

SQL>

SQL> insert into emp\_infovalues(1, 'ruchitavyas', 'web developer', 45000);

insert into emp\_info values(1, 'ruchitavyas', 'web developer', 45000)

\*

ERROR at line 1:

ORA-01722: invalid number

SQL> insert into emp\_infovalues(1, 'ruchitavyas', '101', 45000);

1 row created.

SQL> insert into emp\_infovalues(2, 'heena', '102', 42000);

1 row created.

SQL> insert into emp\_infovalues(3, 'leena', '103', 42050);

1 row created.

SQL> select \* from emp\_info;

EMP\_NO EMP\_NAME EMP\_DEPT EMP\_SALARY

---------- -------------------- ---------- ----------

1 ruchitavyas 101 45000

2 heena 102 42000

3 leena 103 42050

SQL> declare

2 tmp emp\_info%rowtype;

3 no number;

4 cursor c214(no number) is select \* from e\_information where emp\_no = no;

5 begin

6 no:=&no;

7 if not c214%isopen then

8 OPEN c214(no);

9 end if;

10 fetch c214 into tmp;

11 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

12 dbms\_output.put\_line('Employee Details');

13 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

14 dbms\_output.put\_line('EMP\_No: '||'|'||tmp.emp\_no);

15 dbms\_output.put\_line('EMP\_Name: '||'|'||tmp.emp\_name);

16 dbms\_output.put\_line('EMP\_Dept: '||'|'||tmp.emp\_dept);

17 dbms\_output.put\_line('EMP\_Salary:'||'|'||tmp.emp\_salary);

18 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

19 if c214%notfound then

20 dbms\_output.put\_line('Record does not exist');

21 end if;

22

23 if c214%isopen then

24 CLOSE c214;

25 end if;

26 END;

27 /

Enter value for no: 1

old 6: no:=&no;

new 6: no:=1;

cursor c214(no number) is select \* from e\_information where emp\_no = no;

\*

ERROR at line 4:

ORA-06550: line 4, column 42:

PL/SQL: ORA-00942: table or view does not exist

ORA-06550: line 4, column 28:

PL/SQL: SQL Statement ignored

SQL>

SQL> declare

2 tmpemp\_info%rowtype;

3 no number;

4 cursor c214(no number) is select \* from emp\_info where emp\_no = no;

5 begin

6 no:=&no;

7 if not c214%isopen then

8 OPEN c214(no);

9 end if;

10 fetch c214 into tmp;

11 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

12 dbms\_output.put\_line('Employee Details');

13 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

14 dbms\_output.put\_line('EMP\_No: '||'|'||tmp.emp\_no);

15 dbms\_output.put\_line('EMP\_Name: '||'|'||tmp.emp\_name);

16 dbms\_output.put\_line('EMP\_Dept: '||'|'||tmp.emp\_dept);

17 dbms\_output.put\_line('EMP\_Salary:'||'|'||tmp.emp\_salary);

18 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

19 if c214%notfound then

20 dbms\_output.put\_line('Record does not exist');

21 end if;

22 if c214%isopen then

23 CLOSE c214;

24 end if;

25 END;

26 /

Enter value for no: 1

old 6: no:=&no;

new 6: no:=1;

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> declare

2 tmp emp\_info%rowtype;

3 no number;

4 cursor c214(no number) is select \* from emp\_info where emp\_no = no;

5 begin

6 no:=&no;

7 if not c214%isopen then

8 OPEN c214(no);

9 end if;

10 fetch c214 into tmp;

11 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

12 dbms\_output.put\_line('Employee Details');

13 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

14 dbms\_output.put\_line('EMP\_No: '||'|'||tmp.emp\_no);

15 dbms\_output.put\_line('EMP\_Name: '||'|'||tmp.emp\_name);

16 dbms\_output.put\_line('EMP\_Dept: '||'|'||tmp.emp\_dept);

17 dbms\_output.put\_line('EMP\_Salary:'||'|'||tmp.emp\_salary);

18 dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

19 if c214%notfound then

20 dbms\_output.put\_line('Record does not exist');

21 end if;

22 if c214%isopen then

23 CLOSE c214;

24 end if;

25 END;

26 /

Enter value for no: 1

old 6: no:=&no;

new 6: no:=1;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Employee Details

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EMP\_No: |1

EMP\_Name: |ruchitavyas

EMP\_Dept: |101

EMP\_Salary:|45000

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PL/SQL procedure successfully completed.

—-------------------------------------------------------------------------------------------------------

An [explicit cursor](about:blank) may accept a list of parameters. Each time you open the cursor, you can pass different arguments to the cursor, which results in different result sets.

The following shows the syntax of a declaring a cursor with parameters:

CURSOR cursor\_name (parameter\_list)

IS

cursor\_query;

Code language: SQL (Structured Query Language) (sql)

In the cursor query, each parameter in the parameter list can be used anywhere which a constant is used. The cursor parameters cannot be referenced outside of the cursor query.

To open a cursor with parameters, you use the following syntax:

OPEN cursor\_name (value\_list);

Code language: SQL (Structured Query Language) (sql)

In this syntax, you passed arguments corresponding to the parameters of the cursor.

Cursors with parameters are also known as parameterized cursors.

## PL/SQL cursor with parameters example

The following example illustrates how to use a cursor with parameters:

DECLARE

r\_productproducts%rowtype;

CURSOR c\_product (low\_price NUMBER, high\_price NUMBER)

IS

SELECT \*

FROM products

WHERElist\_priceBETWEENlow\_priceANDhigh\_price;

BEGIN

-- show mass products

dbms\_output.put\_line('Mass products: ');

OPEN c\_product(50,100);

LOOP

FETCH c\_product INTO r\_product;

EXIT WHEN c\_product%notfound;

dbms\_output.put\_line(r\_product.product\_name || ': ' ||r\_product.list\_price);

ENDLOOP;

CLOSE c\_product;

-- show luxury products

dbms\_output.put\_line('Luxury products: ');

OPEN c\_product(800,1000);

LOOP

FETCH c\_product INTO r\_product;

EXIT WHEN c\_product%notfound;

dbms\_output.put\_line(r\_product.product\_name || ': ' ||r\_product.list\_price);

ENDLOOP;

CLOSE c\_product;

END;

/

Code language: SQL (Structured Query Language) (sql)

In this example:

* First, declare a cursor that accepts two parameters low price and high price. The cursor retrieves products whose prices are between the low and high prices.
* Second, open the cursor and pass the low and high prices as 50 and 100 respectively. Then fetch each row in the cursor and show the product’s information, and close the cursor.
* Third, open the cursor for the second time but with different arguments, 800 for the low price and 100 for the high price. Then the rest is fetching data, printing out product’s information, and closing the cursor.

## PL/SQL parameterized cursor with default values

A parameterized cursor can have default values for its parameters as shown below:

CURSOR cursor\_name (

parameter\_namedatatype := default\_value,

parameter\_namedatatype := default\_value,

...

) IS

cursor\_query;

Code language: SQL (Structured Query Language) (sql)

If you open the parameterized cursor without passing any argument, the cursor will use the default values for its parameters.

The following example shows how to use a parameterized cursor with default values.

DECLARE

CURSORc\_revenue (in\_yearNUMBER :=2017 , in\_customer\_idNUMBER := 1)

IS

SELECTSUM(quantity \* unit\_price) revenue

FROMorder\_items

INNERJOIN orders USING (order\_id)

WHEREstatus = 'Shipped'ANDEXTRACT(YEARFROMorder\_date) = in\_year

GROUPBYcustomer\_id

HAVINGcustomer\_id = in\_customer\_id;

r\_revenuec\_revenue%rowtype;

BEGIN

OPENc\_revenue;

LOOP

FETCH c\_revenue INTO r\_revenue;

EXIT WHEN c\_revenue%notfound;

-- show the revenue

dbms\_output.put\_line(r\_revenue.revenue);

ENDLOOP;

CLOSE c\_revenue;

END;

Code language: SQL (Structured Query Language) (sql)

In this example, we declared a parameterized cursor with default values. When we opened the cursor, we did not pass any arguments; therefore, the cursor used the default values, 2017 for in\_yearand 1 for in\_customer\_id.

Now, you should know how to use a PL/SQL cursor with parameters to fetch data from the database tables.

## **Note**

* You can only extract YEAR, MONTH, and DAY from a DATE.
* You can only extract TIMEZONE\_HOUR and TIMEZONE\_MINUTE from a timestamp with a time zone datatype.

## **Returns**

The EXTRACT function returns a numeric value when the following parameters are provided: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE\_HOUR, TIMEZONE\_MINUTE, TIMEZONE\_REGION, TIMEZONE\_MINUTE.  
The EXTRACT function returns a VARCHAR2 when TIMEZONE\_REGION or TIMEZONE\_ABBR parameters are provided (because the time zone name or abbreviation information is returned).

## **Applies To**

The EXTRACT function can be used in the following versions of Oracle/PLSQL:

* Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i

## **Example**

Let's look at some Oracle EXTRACT function examples and explore how to use the EXTRACT function in Oracle/PLSQL.

For example:

EXTRACT(YEAR FROM DATE '2003-08-22')

*Result:* 2003

EXTRACT(MONTH FROM DATE '2003-08-22')

*Result:* 8

EXTRACT(DAY FROM DATE '2003-08-22')

*Result:* 22