Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in 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

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK\_ROWCOUNT** and **%BULK\_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **%FOUND**  Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. |
| 2 | **%NOTFOUND**  The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE. |
| 3 | **%ISOPEN**  Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| 4 | **%ROWCOUNT**  Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

Any SQL cursor attribute will be accessed as **sql%attribute\_name** as shown below in the example.

Example

Select \* from cust;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALE |

+----+----------+-----+-----------+----------+

| 1 | Amit. | 34 | Jaipur. | 20000.00 |

| 2 | Ramesh. | 28.| Surat. | 25000.00 |

| 3 | kaushal | 23 | Kota | 29000.00 |

| 4 | Chaitali | 25 | Ajmer | 66500.00 |

| 5 | Harvinde | 27 | Bhopal | 24500.00 |

| 6 | Komal | 22 | MP | 44500.00 |

+----+----------+-----+-----------+----------+

The following program will update the table and increase the sale of each customer by 500 and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected −

DECLARE

total\_rows number(2);

BEGIN

UPDATE customers

SET salary = sale + 500;

IF sql%notfound THEN

dbms\_output.put\_line('no customers selected');

ELSIF sql%found THEN

total\_rows := sql%rowcount;

dbms\_output.put\_line( total\_rows || ' customers selected ');

END IF;

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

6 customers selected

PL/SQL procedure successfully completed.

If you check the records in customers table, you will find that the rows have been updated −

Select \* from cust;

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor 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.

The syntax for creating an explicit cursor is −

CURSOR cursor\_name IS select\_statement;

Working with an explicit cursor includes the following steps −

* Declaring the cursor for initializing the memory
* Opening the cursor for allocating the memory
* Fetching the cursor for retrieving the data
* Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

CURSOR c\_customers IS

SELECT id, name, address FROM cust;

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

OPEN c\_customers;

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

FETCH c\_customers INTO c\_id, c\_name, c\_addr;

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

CLOSE c\_customers;

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

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 cust;

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;

/

PL/SQL procedure successfully completed.

A **record** is a data structure that can hold data items of different kinds. Records consist of different fields, similar to a row of a database table.

For example, you want to keep track of your books in a library. You might want to track the following attributes about each book, such as Title, Author, Subject, Book ID. A record containing a field for each of these items allows treating a BOOK as a logical unit and allows you to organize and represent its information in a better way.

PL/SQL can handle the following types of records −

* Table-based
* Cursor-based records
* User-defined records

Table-Based Records

The %ROWTYPE attribute enables a programmer to create **table-based** and **cursorbased** records.

The following example illustrates the concept of **table-based** records. We will be using the CUSTOMERS table we had created and used in the previous chapters −

DECLARE

customer\_rec cust%rowtype;

BEGIN

SELECT \* into customer\_rec

FROM cust

WHERE id = 5;

dbms\_output.put\_line('Customer ID: ' || customer\_rec.id);

dbms\_output.put\_line('Customer Name: ' || customer\_rec.name);

dbms\_output.put\_line('Customer Address: ' || customer\_rec.address);

dbms\_output.put\_line('Customer Salary: ' || customer\_rec.salary);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Customer ID: 5

Customer Name: Hardik

Customer Address: Bhopal

Customer Salary: 9000

PL/SQL procedure successfully completed.

Cursor-Based Records

The following example illustrates the concept of **cursor-based** records. We will be using the CUSTOMERS table we had created and used in the previous chapters −

DECLARE

CURSOR customer\_cur is

SELECT id, name, address

FROM cust;

customer\_rec customer\_cur%rowtype;

BEGIN

OPEN customer\_cur;

LOOP

FETCH customer\_cur into customer\_rec;

EXIT WHEN customer\_cur%notfound;

DBMS\_OUTPUT.put\_line(customer\_rec.id || ' ' || customer\_rec.name);

END LOOP;

END;

/

PL/SQL procedure successfully completed.

User-Defined Records

PL/SQL provides a user-defined record type that allows you to define the different record structures. These records consist of different fields. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

* Title
* Author
* Subject
* Book ID

Defining a Record

The record type is defined as −

TYPE

type\_name IS RECORD

( field\_name1 datatype1 [NOT NULL] [:= DEFAULT EXPRESSION],

field\_name2 datatype2 [NOT NULL] [:= DEFAULT EXPRESSION],

...

field\_nameN datatypeN [NOT NULL] [:= DEFAULT EXPRESSION);

record-name type\_name;

The Book record is declared in the following way −

DECLARE

TYPE books IS RECORD

(title varchar(50),

author varchar(50),

subject varchar(100),

book\_id number);

book1 books;

book2 books;

Accessing Fields

To access any field of a record, we use the dot **(.)** operator. The member access operator is coded as a period between the record variable name and the field that we wish to access. Following is an example to explain the usage of record −

DECLARE

type books is record

(title varchar(50),

author varchar(50),

subject varchar(100),

book\_id number);

book1 books;

book2 books;

BEGIN

-- Book 1 specification

book1.title := 'C Programming';

book1.author := 'Nuha Ali ';

book1.subject := 'C Programming Tutorial';

book1.book\_id := 6495407;

-- Book 2 specification

book2.title := 'Telecom Billing';

book2.author := 'Zara Ali';

book2.subject := 'Telecom Billing Tutorial';

book2.book\_id := 6495700;

-- Print book 1 record

dbms\_output.put\_line('Book 1 title : '|| book1.title);

dbms\_output.put\_line('Book 1 author : '|| book1.author);

dbms\_output.put\_line('Book 1 subject : '|| book1.subject);

dbms\_output.put\_line('Book 1 book\_id : ' || book1.book\_id);

-- Print book 2 record

dbms\_output.put\_line('Book 2 title : '|| book2.title);

dbms\_output.put\_line('Book 2 author : '|| book2.author);

dbms\_output.put\_line('Book 2 subject : '|| book2.subject);

dbms\_output.put\_line('Book 2 book\_id : '|| book2.book\_id);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book\_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book\_id : 6495700

PL/SQL procedure successfully completed.

Records as Subprogram Parameters

You can pass a record as a subprogram parameter just as you pass any other variable. You can also access the record fields in the same way as you accessed in the above example −

DECLARE

type books is record

(title varchar(50),

author varchar(50),

subject varchar(100),

book\_id number);

book1 books;

book2 books;

PROCEDURE printbook (book books) IS

BEGIN

dbms\_output.put\_line ('Book title : ' || book.title);

dbms\_output.put\_line('Book author : ' || book.author);

dbms\_output.put\_line( 'Book subject : ' || book.subject);

dbms\_output.put\_line( 'Book book\_id : ' || book.book\_id);

END;

BEGIN

-- Book 1 specification

book1.title := 'C Programming';

book1.author := 'Nuha Ali ';

book1.subject := 'C Programming Tutorial';

book1.book\_id := 6495407;

-- Book 2 specification

book2.title := 'Telecom Billing';

book2.author := 'Zara Ali';

book2.subject := 'Telecom Billing Tutorial';

book2.book\_id := 6495700;

-- Use procedure to print book info

printbook(book1);

printbook(book2);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Book title : C Programming

Book author : Nuha Ali

Book subject : C Programming Tutorial

Book book\_id : 6495407

Book title : Telecom Billing

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book\_id : 6495700

PL/SQL procedure successfully completed.