Ex.No. 13	PL/SQL Cursors	Date :

C U R S O R

A cursor is a temporary work area created in the system 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.

There are two types of cursors in PL/SQL. They are Implicit cursors and Explicit cursors.

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

For example, When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected.

When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

Implicit Cursor Attributes

%FOUND

The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row or if SELECT ….INTO statement return at least one row. Ex. SQL%FOUND

%NOTFOUND

The return value is FALSE, if DML statements affect at least one row or if SELECT. …INTO statement return at least one row.
Ex. SQL%NOTFOUND

%ROWCOUNT

Return the number of rows affected by the DML operations
Ex. SQL%ROWCOUNT

Q1) Write a PL/SQL Block, to update salaries of all the employees who work in deptno 20 by 15%. If none of the employee's salary are updated display a message *'None of the salaries were updated'*. Otherwise display the total number

of employee who got salary updated.

```
Declare
  num number(5);
Begin
  update emp set sal = sal + sal*0.15 where deptno=20;
  if SQL%NOTFOUND then
      dbms_output.put_line('none of the salaries were updated');
  elsif SQL%FOUND then
      num := SQL%ROWCOUNT;
      dbms_output.put_line('salaries for ' || num || 'employees are updated');
  end if;
End;
```

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. 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.

There are four steps in using an Explicit Cursor.

- DECLARE the cursor in the declaration section.
- OPEN the cursor in the Execution Section.
- FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

Declaring Cursor : **CURSOR cursor_name IS select_statement;**

Opening Cursor : **OPEN cursor_name;**

Fetching Cursor : **FETCH cursor_name INTO variable-list/record-type;**

Closing Cursor : **CLOSE cursor_name;**

Explicit Cursor Attributes

%FOUND

TRUE, if fetch statement returns at least one row.

Ex. Cursor_name%FOUND

%NOTFOUND

TRUE, , if fetch statement doesn't return a row.

Ex. Cursor_name%NOTFOUND

%ROWCOUNT

The number of rows fetched by the fetch statement.

Ex. Cursor_name%ROWCOUNT

%ISOPEN

TRUE, if the cursor is already open in the program.

Ex. Cursor_name%ISOPEN

Q2) Create a table *emp_grade* with columns *empno & grade*. Write PL/SQL block to insert values into the table *emp_grade* by processing *emp* table with the following constraints.

If sal <= 1400 then grade is 'C'

Else if sal between 1401 and 2000 then the grade is 'B' Else the grade is 'A'.

SQL> create table emp_grade(empno number, grade char(1));

```
Declare     Emp_rec emp%rowtype;
            Cursor c is select * into emp_rec from emp;
Begin
    Open c;
    If  c%ISOPEN  then
        Loop
            Fetch  c  into emp_rec;
            If c%notfound then Exit;  Endif;
            If emp_rec.sal <= 1400 then
               Insert into emp_grade values(emp_rec.empno,'C');
            Elsif emp_rec.sal between 1401 and 2000 then
               Insert into emp_garde values(em_rec.empno,'B');
            Else
               Insert into emp_garde values(em_rec.empno,'A');
            Endif
         End loop;
    Else
        Open c;
    Endif;
End;
```

Ex. No.14	BUILT-IN EXCEPTION	Date:

The PL/SQL block divided into three section: declaration section, the executable section and the exception section

The structure of a typical PL/SQL block is shown in the listing:

Declaration Section :

Defines and initializes the variables and cursor used in the block

Executable commands :

Uses flow-control commands (such as IF command and loops) to execute the commands and assign values to the declared variables

Exception handling :

Provides handling of error conditions

declare
< declaration section >
begin
< executable commands>
exception
<exception handling>
end;

Q1) PROGRAM:

```
declare
a number;
b number;
c number;
begin
a:=&a;
b:=&b;
if(b>0)then
c:=a/b;
dbms_output.put_line('C is'||c);
else
c:=a/b;
end if;
```

```
exception
when zero_divide then
dbms_output.put_line('Divide by zero error');
end;
```

OUTPUT:

```
Enter value for a: 8
Enter value for b: 4
C  is 2

Enter value for a: 4
Enter value for b: 0
Divide by zero error
```

Q2) **PROGRAM:**

```
declare
age number;
inage exception;
begin
age:=&age;
if ((age>=0) and (age<200))then
dbms_output.put_line('Your Age is:'||age);
else
raise inage;
end if;
exception
when inage then
```

```
 dbms_output.put_line('Invalid age error');
 end;
```

OUTPUT:

```
Enter value for age: 20
Your Age is:20

Enter value for age: 0
Invalid age error
```

Ex.No. 15	PL/SQL Triggers	Date :

A trigger is a PL/SQL block structure which is fired when DML statements like Insert, Delete and Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax of Trigger

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
--SQL Statements
END;
```

CREATE [OR REPLACE] TRIGGER trigger_name

This clause creates a trigger with the given name or overwrites an existing trigger with the same name.

BEFORE | AFTER | INSTEAD OF

This clause indicates at what time the trigger should get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. Before and after cannot be used to create a trigger on a view.

INSERT [OR] | UPDATE [OR] | DELETE

This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.

OF col_name

This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.

ON table_name

This clause identifies the name of the table/view to which the trigger is associated.

REFERENCING OLD AS o NEW AS n

This clause is used to reference the old and new values of the data being changed. By default, you reference the values as **:old.column_name** or **:new.column_name**. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

FOR EACH ROW

This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire SQL statement is executed (i.e.statement level Trigger).

WHEN (condition)

This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Types of Triggers

There are two types of triggers based on which level it is triggered.

- *Row level trigger* : An event is triggered for each row updated, inserted or deleted.
- *Statement level trigger* : An event is triggered for each SQL statement executed.

Before and After Triggers : Since triggers occur because of events, they may be set tooccur immediately *before* or *after* those events. Within the trigger, we are able to refer *old* and *new* values involved in transactions. *Old* refers to the data as it existed prior to the transaction. *New* refer to the data that the transaction creates.

Create Table:

SQL> create table employee(eno number(3)primary key,ename varchar(15),dept varchar(5),salary number(5));

Table created.

SQL> desc employee;

Name	Null?	Type
ENO	NOT NULL	NUMBER(3)
ENAME		VARCHAR2(15)
DEPT		VARCHAR2(5)
SALARY		NUMBER(5)

SQL> insert into employee values(&eno,'&ename','&dept',&salary);

Enter value for eno: 101

Enter value for ename: vignesh

Enter value for dept: ca

Enter value for salary: 20000

old 1: insert into employee values(&eno,'&ename','&dept',&salary)

new 1: insert into employee values(101,'vignesh','ca',20000)

1 row created.

SQL> /

Enter value for eno: 102

Enter value for ename: keerthi

Enter value for dept: ca

Enter value for salary: 17000

old 1: insert into employee values(&eno,'&ename','&dept',&salary)

new 1: insert into employee values(102,'keerthi','ca',17000)

1 row created.

SQL> /

Enter value for eno: 103

Enter value for ename: anjali

Enter value for dept: ca

Enter value for salary: 19000

old 1: insert into employee values(&eno,'&ename','&dept',&salary)

new 1: insert into employee values(103,'anjali','ca',19000)

1 row created.

SQL> /

Enter value for eno: 104

Enter value for ename: shaziya

Enter value for dept: ca

Enter value for salary: 16000

old 1: insert into employee values(&eno,'&ename','&dept',&salary)

new 1: insert into employee values(104,'shaziya','ca',16000)

1 row created.

SQL> /

Enter value for eno: 105

Enter value for ename: ravin

Enter value for dept: it

Enter value for salary: 12000

old 1: insert into employee values(&eno,'&ename','&dept',&salary)

new 1: insert into employee values(105,'ravin','it',12000)

1 row created.

SQL> select * from employee;

ENO	ENAME	DEPT	SALARY
101	vignesh	ca	20000
102	keerthi	ca	17000
103	anjali	ca	19000
104	shaziya	ca	16000
105	ravin	it	12000

Q1) PROGRAM:

```
create or replace trigger salarychanges
 before  insert or update on employee
 for each row
 when (new.eno>0)
declare
sal_diff number;
 begin
        sal_diff:=:new.salary - :old.salary;
dbms_output.put_line('Old Salary = '|| :old.salary);
dbms_output.put_line('New Salary = '|| :new.salary);
dbms_output.put_line('Salary Difference = '|| sal_diff);
 end;
 /
```

OUTPUT:

Trigger created.

Triggering a Trigger

1.Insert Command

SQL>insert into employee(eno,ename,dept,salary) values (6,'giri','cs',5000);

Output

Old Salary :

New Salary : 5000

Salary Difference :

2. Update Command

SQL> update employee set salary=salary+500 where eno=102;

Output

Old Salary : 17000

New Salary : 17500

Salary Difference : 500