Experiment No. 11. Regd. No.

Experiment - 11.

Write a PL/SQL code using Cursors, Exceptions and Triggers.

Oracles create a memory area, known as Context area, for processing an SQL statement, which contains all information needed for processing the statement for eg: no 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.

Implicit Cursors:-

Implicit cursors are automatically created by oracle whenver an SQL statement is executed.

Explicit Cursors:-

Explicit cursors are programmer defines cursors for gaining more control over the context area.

Working with an explicit involves four steps:-

Declaring the Cursor for initializing in the memory
Opening the Cursor for allocating memory
Fetching the Cursor for retrieving data.
closing the cursor to release allocated memory.

Experiment No. Regd. No.

Declaring the Cursor

Declaring the cursor, defines the cursor with a name and the associated SELECT statement.

Example: CURSOR c_customers IS
SELECT id, name, address FROM customers;

Opening the cursor:

Opening the cursor allocates memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

For example, we will open above-defined cursor as follows:

→ OPEN c_customers;

Fecthing the Cursor:

Fecthing 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 above opened cursor as follows:

→ CLOSE c_customers.

Experiment No. Regd. No.

```
CREATE PACKAGE Cust_Sal AS
    PROCEDURE find_sal(C_id customers.id%type);
END Cust_Sal;
/
```

The CREATE PACKAGE BOY Statement is used for creating the package body. The following code snippnet shows that Package body declration for the cust-sal package created above.

```
⇒
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';
/
```

Experiment No. Regd. No.

Create a Table with name empl:

```
Create table empl( Empno int, Ename varchar(10), job varchar(9),
sal number(8,2), deptno number(3));
```

Write a program to Copy rows from emp table to empl table using Cursor.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DCLARE
CURSOR emp_cur IS SELECT Empno, Ename, job, sal, deptno
FROM Emp WHERE deptno= &deptno;
cnt number(3);
BEGIN
    FOR emp_rec IN emp_cur
    LOOP
        INSERT INTO empl values emp_rec;
        cnt := emp_cur%ROWCOUNT;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(cnt||' Rows Copied...');
END;
/
```

Output:-

Enter value for deptno: 50

2 Rows Copied.....

PL/SQL procedure successfully completed.

Experiment No. Regd. No.

Exceptions:

What is Exception.

→ An error occur during the program execution is called Expection in PL/SQL.

→ PL/SQL facilitates programmers to catch such Conditions using expection block in the program and an appropriate action is taken against the error Condition.

⇒ There are two types of exceptions.

- System-defined Exceptions.
- User-defined Exceptions.

PL/SQL Exception Handling.

Syntax for exception handling:

Following is a general syntax for exception handling.

1. DECLARE
2. <declaration section>
3. BEGIN
4. <executable Command(s)>
5. EXCEPTION
6. <exception handling goes here>
7. WHEN exception1 THEN
8. exception-1-handling-statements.
9. WHEN exception2 THEN
10. exception2-handling-Statements.
11. WHEN exception3 THEN
12. exception3-handling-Statements.

Experiment No. Regd. No. .

13. ....
14 WHEN others THEN
15. exception3-handling-Statements.
16. END.

Write a pL/SQL program on Exception Handinling.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DCLARE
 emp_rec emp%ROWTYPE;
 BEGIN
    SELECT * INTO emp_rec FROM Emp WHERE empno=&eno;
    DBMS_OUTPUT.PUT_LINE ('Name :'||emp_rec.ename);
    DBMS_OUTPUT.PUT_LINE ('JOb :'|| emp_rec.job);
    DBMS_OUTPUT.PUT_LINE ('Salary :'||emp_rec.sal);
 EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_Line ('NOt Such Record Found..');
        DBMS_OUTPUT.PUT_LINE('SQL Error :'||Sqlerrm);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('SQL Error :'||Sqlerm);
 END;
/
```

Output:-

Enter value for eno. 2022
Name : BALU
Job : ARMY
Salary : 1 Lakh

Experiment No. Regd. No.

Triggers:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.

→ Triggers are stored programs, which are automatically executed or fired when some event occurs.

→ Triggers written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERROR, LOGON, LOGOFF, STARTUP or SHUTDOWN).

Syntax for creating trigger:-

```
1. CREATE [OR REPLACE] TRIGGER trigger_name.
2. { BEFORE | AFTER | INSTEAD OF }
3. { INSERT [OR] | UPDATE [OR] | DELETE }
4. [OF Col_Name]
5. ON table_name
6. [REFERENCING OLD AS o NEW AS n]
7. [FOR EACH ROW]
8. WHEN (condition)
9. DECLARE
10.    declaration-statements.
```

Experiment No. Regd. No. .

11. BEGIN
12. Executable - statements.
13. EXCEPTION
14. Executable - holiding - statements
15. END;

Create 2 Tables Studat and Sudat History Tables.

→ Create Table Studat (Sno number(5), Sname varchar2(12), Course varchar2(10));

→ Create Table Studat History (data varchar2(80));

Write a program to enter details into studat table but not in Sunday.

```
CREATE OR REPLACE TRIGGER holiday
BEFORE INSERT OR DELETE OR UPDATE ON studat:
BEGIN
IF RTRIM(TO_CHAR(SYSDATE,'Day')) = 'Sunday' THEN
        RAISE_APPLICATION_ERROR(-20002,'Today Holiday...');
END IF;
END;
/
```

⇒ SQL> insert into studat values(10,'vinay sri','B.Tech');

Experiment No. Regd. No.

Write a program to store History of details of student Table into student History Table.

```
CREATE OR REPLACE TRIGGER student AFTER INSERT OR UPDATE
OR DELETE ON student FOR EACH ROW
DECLARE
    oper varchar2(12);
BEGIN
    IF INSERTING THEN
        oper:='insertion';
    END IF;
    IF UPDATING THEN
        oper:='updation';
    END IF;
    IF DELETING THEN
        oper:='Deletion'
    END IF;
    INSERT INTO studentHistory
    values('on'||TO_char(sysdate,'DD-MON-YYYY HH24:MM
    :SS')||' '||oper||' Operation took place');
END;
/
```

Experiment No. 12. Regd. No. .

Write a PL/SQL Code using procedures, Function, and packages.

Stored Subprograms:—

A Subprogram is a PL/SQL unit that Consists of SQL and PL/SQL statements that slove a specific problem or perform a set of related tasks.

→ A stored subprogram is a subprogram that is stored in the database.

There are two kind of stored subprograms:

Standalone stored subprogram, which is created at schema level Ex: Stored procedures, Functions.

- Package Subprogram, which is created inside a package Ex: Packages.

What is a stored Procedure?

A stored procedure or in Simple a proc is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

Experiment No. Regd. No.

We can pass parameters to procedure in three ways.

1) IN - parameters
2) OUT - parameters.
3) IN OUT - Parameters.

A procedure may or may not return any value.

General Syntax to create a procedure is;

CREATE [OR REPLACE] PROCEDURE PROC_name [list of parameter]
is
    Declaration section.
    BEGIN
        Execution section
    EXCEPTION
        Execution section.
    END;

Passing parameters to procedure and functions in PL/SQL.

1) IN type parameter: These types of procedures & function are used to send values to store procedures.

2) OUT type parameter: These types of parameters are used to get values from stored procedures.

3) INOUT parameter:- These type of parmeters are used to send values and get values from stored procedure.

Experiment No. [ ] Regd. No. . [ ]

Write a PL/SQL program to create a procedure and perform addition of 2 numbers.

```
CREATE OR REPLACE PROCEDURE addition(x number, y number)IS
r number;
BEGIN
    r:= x+y;
    DBMS_OUTPUT.PUTLINE('Additions:'||r);
END;
/
```

PL/SQL program to be executed on above procedure.

```
SET SERVEROUT ON
SET VERIFY OFF
DCLARE
    a number(3):= &a;
    b number(3):= &b;
BEGIN
    addition(a,b);
END
/
```

Syntax to create a function:

1. CREATE [OR REPLACE] FUNCTION function_name [parameters]
2. [(parameter_name [IN | OUT | IN OUT] type [,...])]
3. RETURN return_datatype
4. [IS | AS]
5. BEGIN
6. < function-body >.
7. END [function_name];

Experiment No. Regd. No. .

Write a PL/SQL program to create a function :-

```
CREATE OR REPLACE FUNCTION sqr (n number) RETURN number is
r number;
BEGIN
    r := n * n;
    return r;
END;
/
```

Packages :=

A package will have two mandatory parts -

- Package specification.
- Package body or definition.

Package Specification:

The specification is the interface to the package. It just DCLARES the types, variables, constants, expections, cursors, and subprograms that can be referenced from outside the package.

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.

Experiment No. Regd. No. .

```
CREATE PACKAGE Cust_Sal AS
    PROCEDURE find_sal(C_id customers.id%type);
END Cust_Sal;
/
```

The CREATE PACKAGE BOY statement is used for creating the package body. The following code snippet shows that package body declaration for the cust_sal package created above.

```
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;
/
```