CHAPTER 9

Module IV

PLSQL

Syllabus

SQL Functions and Procedures in SQL and cursors

9.1 Stored Procedure

Q. Explain Procedures in SQL with suitable example. **MU - Dec. 18, 5 Marks**

- A stored procedure is a named PL/SQL block that can take some parameters (arguments) and produce some output once it is called using stored procedure execution.
- Stored procedures are just text objects, and don't store any data.
- Procedures provides access to data and returns datasets, just like a view.
- Generally speaking you use a procedure to perform a specific action.
- The stored procedure can call data from Table or view.
- Benefits of stored procedures
- Stored procedure offers modularity of code.
- Procedures promote better reusability and maintainability for code.
- Once validated, they can be used in any number of times without compiling again and again in order to make execution faster.
- It can be used in any number of applications to make faster data access.
- If the definition changes only the procedure code is affected. So it will be simple for maintenance.

9.2 Creating Stored Procedure

- A stored procedure has a header, a declaration section, an executable section, and an optional exception-handling section.
- We can create new procedures with help of the CREATE PROCEDURE statement, which may declare a list of parameters, and also must define the actions to be performed by the standard PL/SQL block.
- Stored procedure blocks can start with either BEGIN or the declaration of local variables and end with END statment.
- We cannot reference any host or bind variables inside a stored procedure.
- Using REPLACE option if that procedure exists, it will be dropped and replaced with the new procedure created by the statement.

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode1] datatype1, parameter2 [mode2] datatype2)]
  AS
  BEGIN
  PLSQL CODE;
END;
```

Example

Create a procedure to increment salary of all employees by 10%.

```
SQL> CREATE PROCEDURE spIncrSalary
  AS
  Begin
      UPDATE    Emp
      SET       Sal = 1.1*Sal
      WHERE     Job = 'MANAGER';
  END;
```

- How to create a stored procedure using oracle

a. Enter the text of the CREATE PROCEDURE statement and save it as a script file (*.sql* extension).

b. From SQL*Plus, run the script file to compile the source code.

c. Use SHOW ERRORS command to see any compilation errors.

- How to create a stored procedure using oracle SQL command line

a. Connect to oracle server by providing valid credantials.

b. Write a valid code for stored procedure using above syntax.

c. Use SHOW ERRORS command to see any compilation errors.

9.3 Executing Stored Procedure

- We can execute the stored procedure with help of EXEC command or with help of any application program.
- Executing stored procedure with Oracle

```
SQL>    EXEC spIncrSalary;
```

- Invoking a procedure from PL/SQL Block or from other procedure.

```
DECLARE
....
BEGIN
   spIncrSalary;   -- Invoke procedure from procedure

END;
```

- Invoking a procedure from another procedure

```
SQL>  CREATE  OR  REPLACE  PROCEDURE
access_stud
     IS
     BEGIN
              spIncrSalary;  -- Invoke procedure

     END;
```

9.4 Parameter Types in Stored Procedure

1. Formal parameters

Variables declared in the parameter list of a stored procedure are called as Formal parameters.

Example

```
SQL>  CREATE  PROCEDURE  raise_Fees  (p_id
NUMBER, p_fees NUMBER)
        BEGIN
        ...
        END;
```

2. Actual parameters

Variables or expressions referenced in the parameter list of a stored procedure call are nothing but actual parameters.

Example

```
SQL>    EXEC raise_fees ('1001', 2000);
```

9.5 Parameter Modes in Stored Procedure

1. IN

- This is default mode for any variable declared in PLSQL to pass value to stored procedure.
- Formal parameter acts as constant for that procedure.
- Actual parameter for this type of parameter can be a literal, expression, constant.
- We can assign a default value to IN type of variable.
- Write a procedure to increase 10 % fees of student having given id number.

```
SQL>    CREATE PROCEDURE Raise_Fees
     (v_id IN stud.studno%TYPE)
     IS
     BEGIN
              UPDATE    stud
              SET       fees = fees * 1.10
              WHERE     studno = v_id;
     END;
SQL>    EXEC Raise_Fees (69);
```

2. OUT

- The OUT mode of variable is declared to returns
- a procedure to access student details having given id number.

```
SQL>    CREATE PROCEDURE Query_Stud
     (  p_id IN stud.studno%TYPE,
           p_name OUT stud.name%TYPE,
        p_fees OUT stud.fees%TYPE,
        p_course OUT stud.course%TYPE      )
     IS
     BEGIN
           SELECT name, fees, course
```

```
    INTO p_name, p_fees, p_course
        FROM stud
    WHERE studno = p_id;
END;
```

Executing above stored Procedure

1. Run the above QUERY_EMP procedure.
2. Declare host variables
3. Execute the **QUERY_STUD** procedure and print global variable.

```
SQL > VARIABLE g_name VARCHAR2(20)
SQL > VARIABLE g_fees NUMBER
SQL > VARIABLE g_course NUMBER
SQL > EXECUTE query_emp(25, :g_name, :g_fees, :g_course)
SQL > PRINT g_name
SQL > PRINT g_fees
SQL > PRINT g_course
```

3. IN OUT

- We must specify IN OUT to declare such type of variables.
- Such variable passed into procedure and returns value to calling environment
- These can be initialized and cannot have a default value assigned to it.

Example

Write a procedure to increase 10 % fees of student having given id number.

```
SQL > CREATE PROCEDURE format_Mob_Num
    (p_mobile_no IN OUT VARCHAR2)
    IS
    BEGIN
p_mobile_no := '(91) 0' || SUBSTR(p_ mobile _no,1,4) ||
                '' || SUBSTR(p_ mobile _no,5,7) ||
                '' || SUBSTR(p_ mobile _no,8);
    END;
```

To Executing above stored procedure,

1. Create a host variable using the VARIABLE command.
2. Then populate this host variable with a value, using a PL/SQL block.
3. Invoke the format_Mob_Num procedure supplying the host variable as the IN OUT parameter. We will use of the colon (:) to reference the host variable in the EXECUTE command.
4. To view the value passed back to the calling environment we will use the PRINT command.

```
SQL > VARIABLE g_ mob VARCHAR2(15)
    BEGIN
        :g_ mob := '9821666231';
    END;
    /
SQL > PRINT g_ mob
SQL > EXECUTE Format_Mob_Num (:g_ mob)
SQL > PRINT g_mob
```

9.6 Altering Stored Procedure

1. Introduction

- We can overwrite definition of stored procedure by using ALTER command.
- It also changes procedure details in data dictionary.
- We can change type of parameter in alter option.
- If existence of procedure in not known then we will use option CREATE OR REPLACE which will create new procedure if not existing otherwise replace original procedure.

2. Syntax

```
CREATE OR REPLACE PROCEDURE procedure_name
[(parameter1 [mode1] datatype1, parameter2 [mode2] datatype2)]
AS
BEGIN
 PL/SQL Statements;
END;
```

Example

Write a procedure to increase 20% fees of student having given id number.

```
SQL> CREATE OR REPLACE PROCEDURE Raise_Fees
    (v_id IN stud.studno%TYPE)
    IS
    BEGIN
```

```
    UPDATE      stud
       SET      fees = fees * 1.20
     WHERE      studno = v_id;
END;
```

9.7 Dropping Stored Procedure

1. Introduction

- DROP command is used to remove the procedure from database memory.
- It will remove procedure details from data dictionary also.

2. Syntax

```
DROP PROCEDURE procedure_name
```

3. Example

```
SQL>  DROP PROCEDURE Raise_Fees;
```

9.8 Stored Function

Q. Explain functions in SQL with suitable example. **MU - Dec. 18, 5 Marks**

1. Introduction

- User defined functions are routines that encapsulates SQL logic inside it.
- Like stored procedures User defined functions can also be passed input parameters but user defined functions are compiled and executed at runtime so pretty slower than stored procedures.

Syntax

```
CREATE FUNCTION dbo.myFunction
(
   /*
   @parameter1 datatype = default value,
   @parameter2 datatype
   */
)
RETURNS /* Datatype */
AS
BEGIN
   /* SQL statement ... */
RETURN
   /* value */
END;
```

9.9 Compare stored procedure and Function

Q. Explain the difference between stored procedure and functions in SQL. **MU – Dec. 10, 10 Marks**

Sr. No.	Stored Procedure	Stored Function
1	Invoke as a PLSQL Block of code or using execute statement	Executed as a part of PLSQL or SQL expression.
2	May not return any parameter	Always return parameter
3	It can return one or more output as variables	It must return atleast one variable.
4	It can contain one or more RETURN statements.	It must contain a single RETURN statement.

9.10 Concept of Cursor

Q. Write a short note on : Cursors and its types. **MU - Dec. 17, 5 Marks**

- A **cursor** is a method by which we can assign a name to a result of SELECT statement and manipulate the data within that SQL statement.
- Oracle server uses SQL work areas or result set to execute SQL statements and store its processing information.
- A PL/SQL cursor allows us to name a work area or result set and accesses its stored information.

There are two main types of cursors,

1. Implicit Cursor 2. Explicit Cursor

9.10.1 Implicit Cursor

1. Introduction

- Every SQL DML (Data Manipulation Language) and DRL (Data Retrieval Language - SELECT) statement executed by the Oracle server is having an individual cursor associated with it.

- PL/SQL implicitly declares a cursor for all SQL SELECT and DML statements, including queries that return only one row.
- An implicit cursor is opened and closed by the oracle server automatically to process each SQL statement.
- For example whenever you execute any SELECT statement or any of INSERT, DELETE or UPDATE statement oracle server automatically declares on cursor associate with it.

2. Implicit Cursor Attributes

- An Oracle server implicitly opens and closes such cursors to process each SQL statement.
- Every SQL implicit cursor has many attributes which returns some useful information about the execution of a data manipulation statement or select statement.

Sr. No.	Attribute
1.	SQL%ISOPEN
2.	SQL%FOUND
3.	SQL%NOTFOUND
4.	SQL%ROWCOUNT

SQL%ISOPEN

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will always returns FALSE, as the database closes the SQL cursor automatically after executing its associated SQL statement.

SQL%FOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if an INSERT, UPDATE, or DELETE statement is affecting more then one row or a SELECT INTO statement returned more then one row. Otherwise, it returns FALSE.

SQL%NOTFOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns FALSE if an INSERT, UPDATE, or DELETE statement is affecting more then one row or a SELECT INTO statement returned more then one row. Otherwise, it returns TRUE.

SQL%ROWCOUNT

- This attribute will return integer output.
- This statement will exactly return the number of rows affected by an DML statement, or returned by a SELECT INTO statement.

3. Example

Write a PLSQL block to increment salary of all employees above age 25 in company by 5000 and print number of rows updated by above DML operation.

```
-- Option to Print Output
SQL> SET SERVER OUTPUT ON;

-- Optional declarative section
SQL> DECLARE
 V_number NUMBER(6);
-- Executable section
BEGIN
   UPDATE      Emp
   SET         Sal = Sal + 5000
   WHERE       age > 25;
   V_number:= SQL%ROWCOUNT;
IF SQL%FOUND THEN
 DBMS_OUTPUT.PUT_LINE('Salary of ' ||v_number
|| ' Employees updated.');
ELSE
 DBMS_OUTPUT.PUT_LINE('No Employees having
age above 25');
END IF;
END;

SQL> /
Salary of 20 Employees updated.
PL/SQL procedure successfully completed.
```

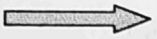
9.10.2 Explicit Cursor

1. Introduction

- We can use explicit cursors to process each row individually returned by a multiple-row SELECT statement in cursor definition.
- Explicit cursor allows us to name a work area and access its stored information.
- Explicit cursors are used in query which returns many rows.

- The set of rows in table fetched by a cursor query is also called active set.
- The size of the active set is rows returned by a select statement.
- Explicit cursor is declared in the DECLARE section of PL/SQL block.
- The Fig. 9.10.1 shows how an explicit cursor points to the current row in the active data set. So program will process only a single row at a time.

Active Set

Current Row 

Employee Table

Eid	Ename	Salary
1	Mahesh	6000
2	Ganesh	4000
3	Suresh	12000

2. Explicit cursor life cycle / cursor implementation

- Inside a PL/SQL block we open a cursor, processes rows returned by a query, and then at last closes the cursor.

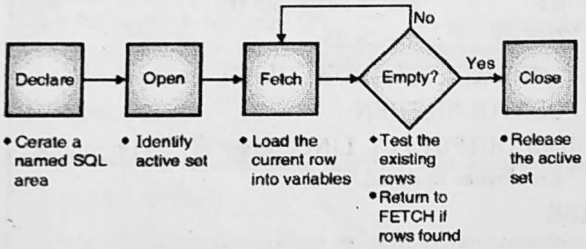


Fig. 9.10.1

a. Declare the cursor

In declare section we defines the structure of the query in cursor.

Syntax

```
CURSOR cursor_name IS select_statement;
```

Example

```
CURSOR     Emp_Cur
IS SELECT  Eid, Ename FROM  Emp;
CURSOR     Employee_Cur
IS SELECT  *          FROM  Emp;
```

b. Open the cursor

- The OPEN statement will execute the query and loads active set.
- After opening cursor, active set is available for fetching.

Syntax

```
OPEN cursor_name;
```

Example

```
OPEN Emp_Cur;
OPEN Employee_Cur;
```

c. Fetch data from the cursor

- After loading data in active set we can fetch row and test the cursor for any existing row.
- We can fetch current row into variables so that it can be accessed and processed.
- If there are no more rows then we need to close the cursor.

Syntax

```
FETCH   cursor_name
INTO    [variable1, variable2, ...] | record_name];
```

- **cursor_name** name of the above declared cursor variable
- **record_name** name of the record in which we retrieved data
- This record variable can be declared with help of %ROWTYPE attribute.
- Include the same number of variables in the INTO clause of as columns returned by SELECT statement, and they should be of compatible datatype. We need to match each variable to correspond to the columns positionally.

Example

```
FETCH Emp_cur      INTO Eid, Ename;
FETCH Employee_cur INTO Employee_Rec;
```

d. Close the cursor

- The CLOSE statement will release the current active set and releases memory occupied by active set.
- It is always possible to reopen the cursor.

Syntax

```
CLOSE cursor_name;
```

Example

```
CLOSE Emp_Cur;
CLOSE Employee_Cur;
```

3. Explicit cursor attributes

- A SQL explicit cursor is opened when we execute OPEN cursor statement.
- Every SQL explicit cursor has many attributes which returns some useful information about cursor state.

Sr. No.	Attribute
1.	Cursor_Name%ISOPEN
2.	Cursor_Name %FOUND
3.	Cursor_Name %NOTFOUND
4.	Cursor_Name %ROWCOUNT

Cursor_Name%ISOPEN

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if cursor OPEN statement is already executed or else it will return value FALSE.
- This statement is used to check cursor is open or not.

Cursor_Name%FOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if cursor active set is having more then one row to be processed after current row. Otherwise, it returns FALSE.
- This statement is used to check all rows in active set are processed or not.

Cursor_Name%NOTFOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns FALSE if cursor active set is having more then one row to be processed after current row. Otherwise, it returns TRUE.
- This statement is used to check all rows in active set are processed or not.

Cursor_Name%ROWCOUNT

- This attribute will return integer output.
- This statement will exactly return the number of rows processed by explicit cursor statement.

4. Example

Write a PLSQL block to print employee name and salary of given employee.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     CURSOR      Emp_Cur
     IS SELECT   Ename, Sal
     FROM        EMP
       WHERE     Empno = &Eid;
                 v_ename  emp.ename%TYPE;
                 v_salary emp.sal%TYPE;

     BEGIN
        IF NOT Emp_Cur%ISOPEN THEN
        OPEN Emp_Cur;
     END IF;
     FETCH Emp_Cur INTO v_ename,v_salary;
     IF Emp_Cur%NOT FOUND THEN
     DBMS_OUTPUT.PUT_LINE('No Such Employee
Found');
     ELSE

   DBMS_OUTPUT.PUT_LINE('Name='||v_ename||'
,salary='||v_salary);
     END IF;
     END;

SQL> /
     Enter value for eid: 7934
     Name=MILLER,salary=1130
     PL/SQL procedure successfully completed.

SQL> /
     Enter value for eid: 10
     No Such Employee Found
     PL/SQL procedure successfully completed.
```

9.10.3 Cursor Basic Loops

1. Introduction

- The simplest form of Iterative statements that can be used to process multiple rows in cursor is a basic loop, which encloses a fetch statement and sequence of statements which is to be repeated between the keywords LOOP and END LOOP.
- Without the EXIT statement, the loop will become infinite loop.

The EXIT statement

- You can terminate a basic loop using the EXIT statement by checking weather it is last row in table.
- Flow of control is passes to the next statement after the END LOOP statement.

2. Syntax

```
LOOP                                    // Start of Loop
   FETCH     cursor_name
   INTO      [variable1, variable2, ...] | record_name];

     Statement 1;                       // Statement

EXIT [WHEN condition]                   // Exit Statement

END LOOP;                               // End of Loop
```

3. Example (Using variables)

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL>    SET SERVEROUTPUT ON;

SQL> DECLARE
      CURSOR    Emp_Cur
      IS SELECT Ename,Sal
      FROM      EMP
      WHERE     age > &Age;
                v_ename   emp.ename%TYPE;
               v_salary  emp.sal%TYPE;
      BEGIN
        IF NOT Emp_Cur%ISOPEN THEN
        OPEN Emp_Cur;
       END IF;
       DBMS_OUTPUT.PUT_LINE('-------------------');
       DBMS_OUTPUT.PUT_LINE('Name    | salary');
       DBMS_OUTPUT.PUT_LINE('-------------------');
       LOOP
       FETCH Emp_Cur INTO v_ename,v_salary;
       EXIT WHEN Emp_Cur%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(RPAD(v_ename,10,' ')||'|
'||v_salary);
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('-------------------');

DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT|
|' rows Selected.');
      DBMS_OUTPUT.PUT_LINE('-------------------');
    END;

SQL> /
    Enter value for age: 25
    Name      salary
    ------------------------------
    SMITH     10800
    ALLEN     11600
    WARD      11250
    JONES     12975
    MARTIN    11250
    BLAKE     12850
    CLARK     12450
    ------------------------------
    7 rows Selected.
    ------------------------------
    PL/SQL procedure successfully completed.
```

4. Example (Using Record set)

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
     DECLARE
          CURSOR Emp_Cur
     IS SELECT Ename,Sal
          FROM  EMP
      WHERE age > &Age;
          Emp_Rec  Emp_Cur %ROWTYPE;

     BEGIN
          IF NOT Emp_Cur%ISOPEN THEN
            OPEN Emp_Cur;
     END IF;
      DBMS_OUTPUT.PUT_LINE('-------------');
      DBMS_OUTPUT.PUT_LINE('Name    | salary');
         DBMS_OUTPUT.PUT_LINE('-------------');
     LOOP
```

```
        FETCH Emp_Cur INTO Emp_Rec;
        EXIT WHEN Emp_Cur%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.enam
  e,10,' ')||'|  '||Emp_Rec.sal);
    END LOOP;
        DBMS_OUTPUT.PUT_LINE('----------------');

  DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOU
NT||' rows Selected.');
        DBMS_OUTPUT.PUT_LINE('----------------');
    END;
SQL> /
  Enter value for age: 25
  Name       Salary
  ------------------------------
  SMITH      10800
  ALLEN      11600
  WARD       11250
  JONES      12975
  MARTIN     11250
  BLAKE      12850
  CLARK      12450
  ------------------------------
  7 rows Selected.
  ------------------------------
  PL/SQL procedure successfully completed.
```

9.10.4 Cursor While Loops

1. Introduction

- We can use the WHILE loop to repeat FETCH operation till the time controlling condition is TRUE as soon as condition becomes FALSE loop will break or exit.
- WHILE condition is always evaluated at the start of each iteration.
- The WHILE loop terminates as soon as the condition is FALSE.

2. Syntax

```
WHILE condition
LOOP                                  // Start of Loop
FETCH cursor_name
INTO [variable1, variable2] | record_name];
   Statement 1;                       // Statement
   ....
END LOOP;                             // End of Loop
```

Where,

Condition - This condition is evaluated at the starting of each iteration; this loop will be repeated while a condition is TRUE.

3. Example

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
      CURSOR        Emp_Cur
      IS SELECT Ename,Sal
      FROM          EMP
      WHERE         age > &Age;
                 Emp_Rec  Emp_Cur%ROWTYPE;

    BEGIN
      IF NOT Emp_Cur%ISOPEN THEN
        OPEN Emp_Cur;
    END IF;
      DBMS_OUTPUT.PUT_LINE('----------------------');
      DBMS_OUTPUT.PUT_LINE('Name      | salary');
      DBMS_OUTPUT.PUT_LINE('----------------------');
    WHILE Emp_Cur%FOUND
    LOOP
      FETCH Emp_Cur INTO Emp_Rec;

  DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.enam
e,10,' ')||'|  '||Emp_Rec.sal);
    END LOOP;
      DBMS_OUTPUT.PUT_LINE('----------------------');

  DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOU
NT||' rows Selected.');
      DBMS_OUTPUT.PUT_LINE('----------------------');
    END;
SQL> /
    Enter value for age: 25
    Name       Salary
    ------------------------------
```

```
------------------------------
SMITH     10800
ALLEN     11600
WARD      11250
JONES     12975
MARTIN    11250
BLAKE     12850
CLARK     12450
------------------------------
7 rows Selected.
------------------------------
PL/SQL procedure successfully completed.
```

9.11 Cursor FOR Loops

1. Introduction

- FOR loop will have a control statement before LOOP keyword which determine the number of rows fetched by a cursor.
- In case of FOR loop there is no need to declare the record variable in code; it is declared implicitly as a cursor_name%ROWTYPE by system itself.
- We can refer the record variable inside loop only; we can use an expression to reference the existing value of this record set variable.

2. Syntax

```
FOR record_name IN cursor_name
LOOP                          // Start of Loop
    Statement 1;              // Statement
    ....
END LOOP;                     // End of Loop
```

- FOR Loop is shortcut for above type of loops as there is no need to OPEN, CLOSE and FETCH cursor explicitly it will be done automatically.
- Even Record set variable is declared automatically.
- No need to write terminating condition as soon as cursor reaches to last row it will exit from fetch operations and closes cursor automatically.

3. Example

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     CURSOR      Emp_Cur
     IS SELECT   Ename,Sal
     FROM        EMP
     WHERE       age > &Age;
     BEGIN
      DBMS_OUTPUT.PUT_LINE('----------------');
      DBMS_OUTPUT.PUT_LINE('Name | salary');
      DBMS_OUTPUT.PUT_LINE('----------------');
     FOR Emp_Rec IN Emp_Cur
     LOOP

DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,1
0,' ')||'|  '||Emp_Rec.sal);
     END LOOP;
      DBMS_OUTPUT.PUT_LINE('----------------');

DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT|
|' rows Selected.');
      DBMS_OUTPUT.PUT_LINE('----------------');
    END;
SQL> /
   Enter value for age: 25
   Name      Salary
   ------------------------------
   SMITH     10800
   ALLEN     11600
   WARD      11250
   JONES     12975
   MARTIN    11250
   BLAKE     12850
   CLARK     12450
   ------------------------------
   7 rows Selected.
   ------------------------------
   PL/SQL procedure successfully completed.
```

4. Cursor FOR loops with subquery

- In FOR loop we can declare subquery in place of cursor variable.
- So any query that is solved by cursor can be solved without use of cursor by using cursor query in FOR loop.

5. Syntax

```
FOR record_name IN (SubQuery)
LOOP                    -- Start of Loop
   Statement 1;         -- Statement
   ....
END LOOP;               -- End of Loop
```

- FOR Loop is shortcut for above type of loops as there is no need to OPEN, CLOSE and FETCH cursor explicitly it will be done automatically.
- Even Record set variable is declared automatically.
- No need to write terminating condition as soon as subquery reaches to last row it will exit from fetch operations.

6. Example

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
      DBMS_OUTPUT.PUT_LINE('------------------');
      DBMS_OUTPUT.PUT_LINE('Name    | salary');
      DBMS_OUTPUT.PUT_LINE('----------------');
     FOR Emp_Rec IN (SELECT Ename,Sal FROM
     EMP WHERE age>&Age)
     LOOP
DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,1
0,' ')||'| '||Emp_Rec.sal);
     END LOOP;
      DBMS_OUTPUT.PUT_LINE('------------------');
     END;
Output
SQL> /
  Enter value for age: 25
  Name  Salary
  ------------------------------
  SMITH     10800
  ALLEN     11600
  WARD      11250
  JONES     12975
  MARTIN    11250
  BLAKE     12850
  CLARK     12450
  ------------------------------
  PL/SQL procedure successfully completed.
```

9.12 Parameterized Cursor

1. Introduction

- Sometime values for cursor declaration need to pass at runtime such values can be passed while opening cursor using parameter in cursor.
- Parameters allow values to be passed to a cursor while opening it and to be used in the execution of query.
- That is we can open and close an explicit cursor several times in a PLSQL block and return different active set each time.
- Each of formal parameter in the cursor declaration must have a corresponding actual parameter at the time of OPEN cursor.
- Parameter data types should be same as scalar variables, but there is no need to give them sizes.
- The parameter names are just for references in the query expression of the cursor.

2. Syntax

```
CURSOR cursor_name [(parameter_name datatype)]
IS
      select_statement;
```

- Cursor_name is a PL/SQL identifier for cursor.
- Parameter_name is the name of a parameter to be passed.
- Datatype is a scalar datatype of the parameter without length.
- Select_statement is a SELECT Query statement

3. Example

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     CURSOR        Emp_Cur (p_age NUMBER)
     IS SELECT  Ename,Sal
       FROM        EMP
       WHERE  age > p_age;
                   Emp_Rec
Emp_Cur%ROWTYPE;

    BEGIN
     IF NOT Emp_Cur%ISOPEN THEN
```

```
    OPEN        Emp_Cur(&Age);
    END IF;
      DBMS_OUTPUT.PUT_LINE('------------------');
      DBMS_OUTPUT.PUT_LINE('Name                |
salary');
      DBMS_OUTPUT.PUT_LINE('------------------');
    LOOP
    FETCH       Emp_Cur INTO Emp_Rec;
    EXIT WHEN Emp_Cur%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,1
0,' ')||'|  '||Emp_Rec.sal);
    END LOOP;
      DBMS_OUTPUT.PUT_LINE('--------------------');

DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT|
|' rows Selected.');
      DBMS_OUTPUT.PUT_LINE('------------------');
   END;

SQL> /
   Enter value for age : 25
   Name      Salary
   ------------------------------
   SMITH     10800
   ALLEN     11600
   WARD      11250
   JONES     12975
   MARTIN        11250
   BLAKE         12850
   CLARK         12450
   ------------------------------
   7 rows Selected.
   ------------------------------
   PL/SQL procedure successfully completed.
```

9.13 Cursor Variables

1. Introduction

- Cursor variables are like C pointers, which hold only memory location (address) instead of the item itself. So, declaring a cursor variable creates only a pointer and not an item.
- In PL/SQL, a pointer has datatype as,

REF X

where REF = REFERENCE

X = class of objects.

- Hence, a cursor variable has datatype called as a REF CURSOR.
- We use an explicit cursor which names the work area or we can also use a cursor variable which points to the work area.
- A cursor always refers to the same work area. But a cursor variable can refer to different work areas. So both are operationally different
- We use cursor variables to pass query result sets between various PL/SQL stored subprograms and clients. So they only shears a pointer to the query work area in which the result set is stored.
- A query work area accessible till the time cursor variable points to it. Therefore, we can pass the value of a cursor variable from one scope to another easily.
- Cursor variables are available for use to every client in database environment.
- We can pass cursor variables from application software to the database server using remote procedure calls.

2. Syntax

- Define a REF CURSOR type
- TYPE **ref_type_name** IS REF CURSOR
- [RETURN return_type];
- Declare a cursor variable of that type
- ref_cv ref_type_name

3. Example

Write a PLSQL block to print employee name and salary of all employees.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     TYPE    Emp_Ref_Cur IS REF CURSOR;
             Emp_Cur Emp_Ref_Cur;
     TYPE rec_emp is record
     (
       name varchar2(20),
       sal  number(6)
     );
```

```
    Emp_Rec rec_emp;
    BEGIN
     OPEN Emp_Cur for select ename,sal from emp;
     DBMS_OUTPUT.PUT_LINE('------------------');
     DBMS_OUTPUT.PUT_LINE('Name      | salary');
     DBMS_OUTPUT.PUT_LINE('---------------------');
     LOOP
        fetch Emp_Cur into Emp_Rec;
        exit when Emp_Cur%notfound;

DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.name,10
,')||'|  '||Emp_Rec.sal);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('---------------------');
    CLOSE EMP_CUR;
     END;
SQL> /
   Name          Salary
   ---------------------------------
   SMITH         10800
   ALLEN         11600
   WARD          11250
   JONES         12975
   MARTIN        11250
   BLAKE         12850
   CLARK         12450
   ---------------------------------
   7 rows Selected.
   ---------------------------------
   PL/SQL procedure successfully completed.
```

9.14 Sample Programs

Write a PLSQL block to print employee name and salary and show salary increment of 10% of all employees crossing age of 60.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
     CURSOR     emp_cur
     IS
      SELECT    empno, sal
      FROM      emp
      WHERE     age > 60;
          v_empno emp.empno%TYPE;
          v_sal emp.sal%TYPE;
     BEGIN
     OPEN           emp_cur;
        DBMS_OUTPUT.PUT_LINE('Empno'||'
'||'sal'||'  '||'Incr. salary');
     LOOP
     FETCH    emp_cur INTO v_empno,v_sal;
     EXIT WHEN emp_cur%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(v_empno||'
'||v_sal||'   '||1.1*v_sal);
     END LOOP;

DBMS_OUTPUT.PUT_LINE(emp_cur%ROWCOUNT|
|' Rows selected..!');
     CLOSE emp_cur;
     END;
SQL> /
   Empno      Sal      Incr. salary
   7369       10800        11880
   7499       11600        12760
   7521       11250        12375
   7566       12975        14272.5
   7654       11250        12375
   7698       12850        14135
   7782       12450        13695
   7788       13000        14300
   7839       15000        16500
   7844       11500        12650
```

Write a PLSQL block to print all employees name with last digit of age is 6.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL>   DECLARE
       CURSOR     Emp_Cur
       IS         SELECT Ename, Sal
          FROM       EMP;
       BEGIN
            DBMS_OUTPUT.PUT_LINE('Employee
Name');
       FOR Emp_Rec IN Emp_Cur
       LOOP
```

```
        IF        Emp_Rec.age -
trunc(Emp_Rec.age,-1) = 6
        THEN
  DBMS_OUTPUT.PUT_LINE(Emp_Rec.ename);
        END IF;
        END LOOP;
    END;
```

Write a PLSQL BLOCK to accept radius of circle from user and insert radius and area in area table.

```
SQL> Create table area (radius Number (7, 2),Area
Number(7,2));
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     v_radius area.radius%TYPE:=&Radius;
     v_area  area.area%TYPE;
     BEGIN
      v_area := 3.14 * v_radius * v_radius;
     INSERT INTO Area(radius,area)
     VALUES (v_radius,v_area);
     IF SQL%found THEN
     DBMS_OUTPUT.PUT_LINE('1 row Inserted.');
     END IF;
     COMMIT;
     END;
```

Write implicit cursor to count number of rows updated by update statement.

```
-- The following parameter is required for printing
DBMS_OUTPUT.
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
     Rows_affected CHAR(4);
     BEGIN
      UPDATE  emp
      SET     sal = sal + 1000;
              rows_affected  :=  to_char(SQL  %
              rowcount);
      IF SQL % rowcount > 0 THEN

DBMS_OUTPUT.PUT_LINE(rows_affected);
      ELSE
     DBMS_OUTPUT.PUT_LINE('NO          ROWS
     AFFECTED');
     END IF;
     END;
```

Review Questions

1. Write note on User Define Function.
2. Write short note on stored procedure
3. Write short note on implementation of stored procedure
4. Explain how to pass parameters of stored procedures
5. Explain how we can change definition of stored procedures
6. How to drop stored procedures
7. Explain how we can change definition of stored procedures.
8. What is PLSQL Cursors ?
9. Explain various types of cursors with example.
10. Explain Implicit cursor attributes in details with example.
11. Explain cursor attributes in details with example.
12. Explain cursor for loops in details with example.
13. Write short notes on :
    a. Implicit Cursor
    b. Cursor For Loops
    c. Cursor Loops
14. What are the parameterized cursors in PLSQL ?
15. Explain cursor variables in details.
16. Explain REF Cursors in details.
17. Crete a functions which returns the total number of records in Employee table.

9.15 University Questions And Answers

Dec. 2017

1. Explain the difference between stored procedure and functions in SQL. **(10 Marks)**
2. Write a short note on : Cursors and its types **(5 Marks)**

Dec. 2018

3. Explain Functions and Procedures in SQL with suitable example. **(10 Marks)**

□□□