Swami Sahajanand College of Computer Science

B.C.A. SEM-VI

Subject: RDBMS using Oracle-II

UNIT 1

Basic PL/SQL Programming

- PL/SQL Block Structure.
- Control Structure.
- Implicit Cursor Programming.
- Explicit Cursor Programming.
- Parameterize Cursor and Cursor For Loop.

- **PL/SQL Block Structure**
- **Introduction of PL/SQL:**
- PL/SQL stands for Procedural Language / Structured Query Language.
- PL/SQL is an extension of SQL.
- It is a procedural language.
- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- PL/SQL is specially designed for database oriented activities.
- PL/SQL is a completely portable, high-performance transaction-processing language. PL/SQL provides a built-in, interpreted and OS independent programming environment. PL/SQL can also directly be called from the command-line SQL*Plus interface.
- We can use PL/SQL to implement our business application by creating stored function, cursor and triggers.
- PL/SQL code is grouped into structures called as blocks.
- With the PL/SQL language, we can create schema objects, including following…
  - **Stored procedures & functions:**
  - A stored procedure is a PL/SQL program that can be enabled by an application, a trigger, or an Oracle tool.
  - Difference between procedure & function is that function returns a result but procedure cant.
  - **Packages:**
  - A package is a file that groups functions, cursors, stored procedures and variables in one place.
  - **Triggers:**
  - A trigger is a PL/SQL program that is stored in the database and executed immediately before or after the INSERT, UPDATE and DELETE commands.
  - **Cursors:**
  - Oracle uses workspaces to execute the SQL commands.
  - Through PL/SQL cursors, it is possible to name the workspace and access its information.
- **Advantages of PL/SQL:**
- **PL/SQL supports both static and dynamic SQL:**
- Static SQL supports DML operations and transaction control from PL/SQL block.
- In Dynamic SQL support DDL operations and transaction control from PL/SQL block.
- **Reduce network traffic:**
- PL/SQL allows sending an entire block of statements to the database at one time.
- This reduces network traffic and provides high performance for the applications.
- **Procedural language support:**
- PL/SQL is a development tools not only for data manipulation futures but also provide the conditional checking, looping or branching operations same as like other programming language.
- **Error handling:**
- PL/SQL is dealing with error handling, its permits the smart way handling the errors and giving user friendly error messages, when the errors are occur.
- **Declare variable:**
- PL/SQL gives you control to declare variables and access them within the block.
- The declared variables can be used at the time of query processing.

- **Intermediate Calculation:**
- Calculations in PL/SQL done quickly and efficiently without using Oracle engines.
- This improves the performance of transaction.
- **Portable application:**
- Applications are written in PL/SQL are portable in any Operating system.
- PL/SQL applications are independence program to run any computer.

- **Features of PL/SQL:**

(1)Block structure:
- PL/SQL is a block-structured language. Each program written in PL/SQL is written as a block.
- Blocks can also be nested. Each block is meant for a particular task.

(2)Variables and constants:
- PL/SQL allows you to declare variables and constants.
- Variables are used to store values temporarily.
- Variables and constants can be used in SQL and PL/SQL procedural statements just like an expression.

(3)Control structures:
- PL/SQL allows control structures like IF statement, FOR loop, WHILE loop to be used in the block.
- Control structures are most important features in PL/SQL.

(4)Exception handling:
- An errors is to be detected and it is solve using exception.
- Whenever there is a predefined error PL/SQL raises an exception automatically.
- These exceptions can be handled to recover from errors.

(5)Modularity:
- PL/SQL allows process to be divided into different modules.
- Subprograms called as procedures and functions can be defined and invoked using the name.
- These subprograms can also take parameters.

(6)Cursors:
- A cursor is a private SQL area used to execute SQL statements and store processing information.
- PL/SQL implicitly uses cursors for all DML commands and SELECT command that returns only one row. And it also allows you to define explicit cursor to deal with multiple row queries.

(7)Built-in functions:
- Most of the SQL functions that we have seen so far in SQL are available in PL/SQL.
- These functions can be used to manipulate variables of PL/SQL.

- **A Simple PL/SQL Block structure:**
- Each PL/SQL program consists of SQL and PL/SQL statements which makes a PL/SQL block. A PL/SQL Block consists of three sections:

  - **The Declaration section (optional).**
  - **The Execution section (mandatory).**
  - **The Exception (or Error) Handling section (optional).**

- **Declaration Section:**
- The Declaration section of a PL/SQL Block starts with the reserved keyword **DECLARE**.
- This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section.
- Placeholders may be any of Variables, Constants and Records, which stores data temporarily.
- Cursors are also declared in this section.
- **Execution Section:**
- The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END.
- This is a mandatory section and is the section where the program logic is written to perform any task.
- The programmatic constructs like loops, conditional statement and SQL statements are written into execution section.
- **Exception Section:**
- The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION.
- This section is optional.
- Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully.
- If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.
  - *Every statement in the above three sections must end with a semicolon;*
  - *PL/SQL blocks can be nested within other PL/SQL blocks.*
  - *Comments can be used to document code*.

- **Simple PL/SQL Block Structure:**

DECLARE section (Optional)

Declaration of memory variables, Variable, Cursors, User-Defined Exception, constants,

BEGIN section (Mandatory)

SQL executable statements
PL/SQL executable statements.

EXCEPTION section (Optional)

SQL or PL/SQL code to handle errors that may arise during the execution of the code block between BEGIN and EXCEPTION section.

END; (Mandatory)

Each & every PL/SQL block terminates with END section.

Command to Create PL/SQL block.

- **Syntax:**
  Edit program_name.sql
- **Example:**
  Edit first.sql

Command to execute PL/SQL block

- **Syntax:** Start program_name.sql
  OR
  @ program_name.sql
- **Example:** Start first.sql
  OR
  @ first.sql

Example: PL/SQL block Program

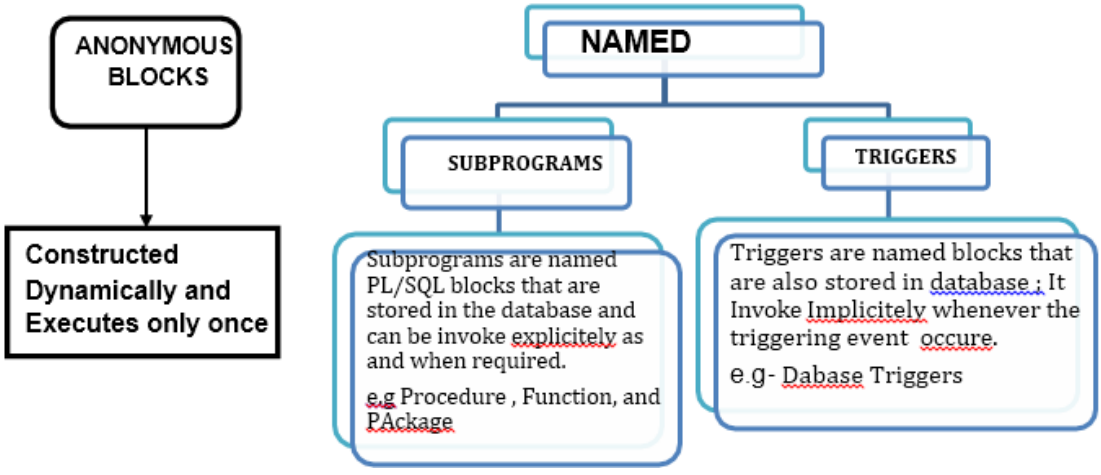
```
Edit first.sql
   SETserveroutput on
   BEGIN

            dbms_output.put_line('Welcome to Oracle');

   END;
```

Output:

Welcome to Oracle

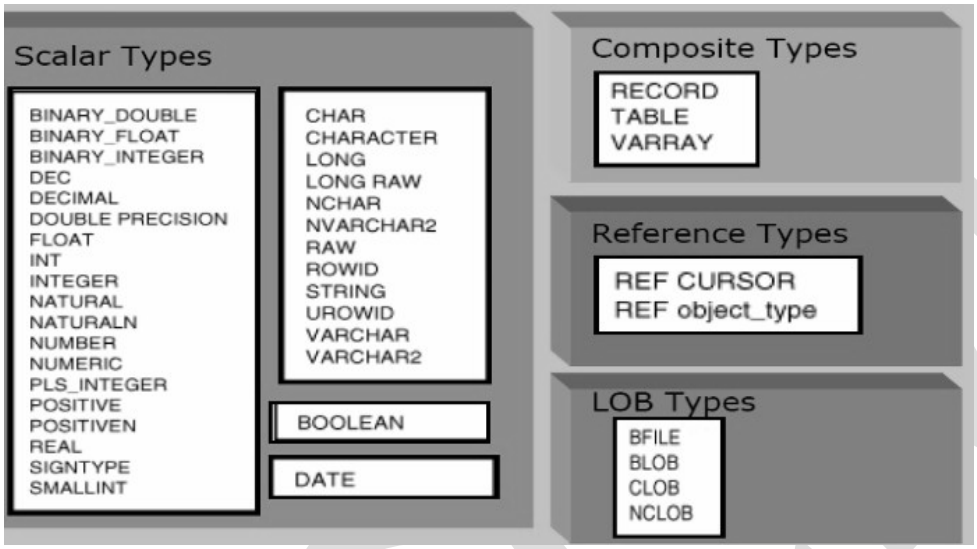
- **SET SERVEROUTPUT ON**
- By default it is off.
- To display any message on the oracle screen we have to first set **serveroutput** on.
- **dbms_output.put_line()**
- This function is used display the message as well as value of variable.
- dbms_output is a package that includes a number of procedures and functions that collect information in a buffer so it can be retrieved later.
- put_line puts a piece of information in the package buffer followed by an end-of-line marker.

- **What are the various types of PL/SQL blocks?**
- PL/SQL is a block structured language.
- It is composed of one or more blocks.



- **PL/SQL :-Data Types and Subtypes**

  "Data types means the type of value a variable can stored."
- **Number** – Store numeric values on which arithmetic operations are performed.
- **Char**- Store Alphanumeric values that represent single characters or strings of characters.
- **Date**- It is used to store Date and time type of value.
- **Boolean** - Logical values on which logical operations are performed (True/False).



- **%TYPE:**
- %TYPE is used for retrieving column value from table.
- PL/SQL can use the %TYPE attribute to declare variables based on definitions of columns in a table.
- If, a column's attribute change, the variable's attributes will change as well.
- %TYPE declares a variable or constant to have the same data type as that of a previously defined variable or of a column in a table or in a view.
- **Example: STUDENT.S_ID % type;**
- Here, STUDENT is table name and S_ID is field name (Column name).
- When referencing a table, a user may name the table and the column, or name the owner, the table and the column.
- **Declaring Variables:**
- Variables can have any data type, such as CHAR, DATE, NUMBER, BOOLEAN, etc.
- For example, assume that you want to declare variables for part Employee no, such as empno to hold 6-digit numbers and gender to hold the Boolean value TRUE or FALSE.
- Note that there is a semi-colon (;) at the end of each line in the declaration section. You can assign a value to a variable when it is declared.

```
DECLARE
    empnoNUMBER(6);
    empname VARCHAR2(20);
    gender BOOLEAN;
    hourly_salary NUMBER := 22.50;
    bonus  NUMBER := 150;
```

```
DECLARE
  num1 INTEGER;
  num2 REAL;
  num3 DOUBLEPRECISION;
  BEGIN
       null;
  END;
```

- **Control Structure:**
- Following are three categories of control structure in PL/SQL.
  - **Conditional Control (If statement).**
  - **Iterative Control (Looping Structure).**
- **Conditional Control (If statement).**
- Conditional control is used for decision- making.
- Decision-making structures require that the programmer specify one or more conditions to be evaluated by the program.
- Statement are executed according to condition. There are following type of conditional statement:
  - **IF THEN Statement**
  - **IF THEN ELSE Statement**
  - **IF THEN ELSIF Statement (Else-if leader)**
  - **Nested IF THEN ELSE Statement**
  - **CASE Statement**
- **IF THEN Statement:**
- When an IF-THEN statement is executed, a condition is evaluated to either TRUE or FALSE.
- If the condition is TRUE, control is passed to the first executable statement of the IF-THEN construct.
- If the condition is FALSE, control is passed to the first executable statement after the END IF statement.

:: Syntax ::

```
IF (condition) THEN
   statement
END IF;
```

:: Output ::

Condition true

:: Example ::

```
DECLARE
  no INTEGER(2) := 10;
  BEGIN
  IF (no = 10) THEN
    DBMS_OUTPUT.PUT_LINE('Condition true');
  END IF;
END;
```

- **IF THEN ELSE Statement:**
- IF condition is TRUE then execute statement-1 part and cursor move next to END IF.
- IF condition is FALSE then execute statement-2 part and cursor move next to END IF.

:: Syntax ::

```
IF (condition) THEN
   Statement-1
ELSE
  Statement-2
END IF;
```

:: Example ::

```
DECLARE
  no INTEGER(2) := 10;
BEGIN
  IF (no = 10) THEN
      DBMS_OUTPUT.PUT_LINE('condition TRUE');
  ELSE
      DBMS_OUTPUT.PUT_LINE('condition FALSE');
  END IF;
END;
```

3) **IF THEN ELSIF Statement:**

- If you want to execute only one statement block out of multiple condition then it is used.
- Here we give another condition in ELSE part.
- We used ELSIF for another condition.
- The words CONDITION 1 through CONDITION N are a sequence of the conditions that evaluate to TRUE or FALSE. These conditions are mutually exclusive.
- In other words, if CONDITION 1 evaluates to TRUE, STATEMENT 1 is executed, and control is passed to the first executable statement after the reserved phrase END IF.
- The rest of the ELSIF construct is ignored. When CONDITION 1 evaluates to FALSE, control is passed to the ELSIF part and CONDITION 2 is evaluated, and so on.

:: Syntax ::

```
IF (condition-1) THEN
   Statement-1
ELSIF (Condition-2) THEN
  Statement-2
ELSIF (Conition-3) THEN
  Statement-3
ELSE
  Statement-N
END IF;
```

:: Output ::

```
No is 10
```

:: Example ::

```
DECLARE
  no INTEGER(2) := 10;
BEGIN
  IF (no = 5) THEN
        DBMS_OUTPUT.PUT_LINE ('No is 5');
  ELSIF (no=10) THEN
        DBMS_OUTPUT.PUT_LINE ('No is 10');
  ELSIF (no=15) THEN
        DBMS_OUTPUT.PUT_LINE ('No is 15');
   ELSE
         DBMS_OUTPUT.PUT_LINE ('No is'||no);
  END IF;
END;
```

4) **Nested IF THEN ELSE Statement:**

- If we want to use IF THEN ELSE inside another IF THEN ELSE statement then nested IF is used.
- We write IF condition into another IF statement block.
- According to condition particular block will be executed.

:: Syntax ::

```
IF (condition-1) THEN
        IF (condition-2)
              THEN
              Statement-1
        ELSE
              Statement-2
        END IF;
ELSE
 Statement-3
END IF;
```

:: Output ::

```
No is 10
```

:: Example ::

```
DECLARE
  no INTEGER(2) := 10;
   BEGIN
  IF (no > 5) THEN
        IF (no=10) THEN
           DBMS_OUTPUT.PUT_LINE ('No is 10');
        ELSE
           DBMS_OUTPUT.PUT_LINE ('No is not 10');
        END IF;
ELSE
    DBMS_OUTPUT.PUT_LINE ('No is'||no);
  END IF;
END;
```

- **CASE Statement:**
- If we require multiple IF then we used CASE statement.
- CASE statement selects one sequence of statement to execute.
- However, to select the sequence, the CASE statement uses selector rather than multiple Boolean expression.
- A selector is an expression, the value of which is used to select one of several alternatives.

:: Syntax ::

```
CASE selector
   WHEN value-1 THEN
      Statement-1;
   WHEN value-2 THEN
      Statement-2;
   ELSE
      Statement-3;
END CASE;
```

:: Output ::

```
value 3
```

:: Example ::

```
DECLARE
  no INTEGER := 3; BEGIN
   CASE no
      WHEN 1 THEN
         DBMS_OUTPUT.PUT_LINE ('value 1');
      WHEN 2 THEN
         DBMS_OUTPUT.PUT_LINE ('value 2');
      WHEN 3 THEN
         DBMS_OUTPUT.PUT_LINE ('value 3');
      ELSE
         DBMS_OUTPUT.PUT_LINE ('no matching CASE found');
   END CASE;
END;
```

- **<u>Iterative Control (Looping Structure).</u>**
- Looping structure is used to reduce number of common statements in a program for writing repeatedly.
- It means sometimes we want to execute certain statements to a fix number of times at that time we write those statements repeatedly then the code becomes too long which is not a good programming practice.
- Looping is a process that executes some of the common statements repeatedly up to finite number of times (Statement write only one time).
- In looping, set of statements are executed until some conditions for termination of the loop is satisfied.
- A program loop therefore consists of two parts one known as body of the loop and other is known
- as the test condition statement.
- The test condition statement checks certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- Following looping structure available in PL/SQL:
    - Simple LOOP
    - WHILE LOOP
    - FOR LOOP

(1) Simple LOOP:

- In this loop the sequence of statements encloses in between LOOP and END LOOP statement.
- When each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

- Simple loop is also known as exit controlled loop.
- In this loop condition is check at the end of loop.
- Here, sequence of statements may be single statement or multiple statement.
- An EXIT statement or EXIT WHEN statement is required to break the loop.

:: Syntax ::

```
LOOP
        Statement-1
        Statement-2
   EXIT WHEN Condition;
END LOOP;
```

:: Output ::

1 2 3 4 5

:: Example ::

```
DECLARE
      i number (3);
BEGIN
      i:=1;
      loop
            dbms_output.put(i);
            i:=i+1;
         exit when i>5;
      end loop;

END;
```

(2) WHILE LOOP

- The reserved word WHILE marks the beginning of a loop construct.
- In this loop the condition is check first.
- If condition is true then only statement will be executed.
- Whenever condition is false then cursor move to the next executable statement after END LOOP.
- Statements 1 through N are a sequence of statements that is executed repeatedly.
- The END LOOP is a used to show the end of the loop.

:: Syntax ::

```
WHILE <condition>
   LOOP
      Statement-1
      Statement-2
      ……..
      Statement- N
   END LOOP;
```

:: Output :: 1 2 3 4 5

:: Example ::

```
DECLARE
      i number (3);
BEGIN
      i:=1;
      WHILE (i<=5)
      loop
            dbms_output.put(i);
            i:=i+1;
      end loop;
END;
```

(3) FOR LOOP:

- For loop is considered as Counter Loop.
- For loop is used when series of statement needs to be executed for specific number of times.
- For loop is an entry control loop.

- For loop is implemented on Number Datatype only.
- Scope of Variable is Private and it is within Loop.
- First we need to initialize, then need to perform increment / decrement.
- REVERSE Key word is used to execute Loop in Reverse order (Ex:- Print 10 to 1)
- For loop is execute as per the range you defined in the loop.
    - Variable in for loop need not be declared.
    - Increment value cannot be specified.
    - For loop variable is always incremented by 1.

:: Syntax ::

```
For <variable-name> in [REVERSE] <start-value>..<Stop-value>

Loop
        Statements;
End loop;
```

:: Example ::

```
DECLARE
    I number (3);
BEGIN
    For I in 1..5
    loop
      dbms_output.put(i);
    end
loop;
END;
```

Implicit Cursor Programming

- **What is Cursor?**

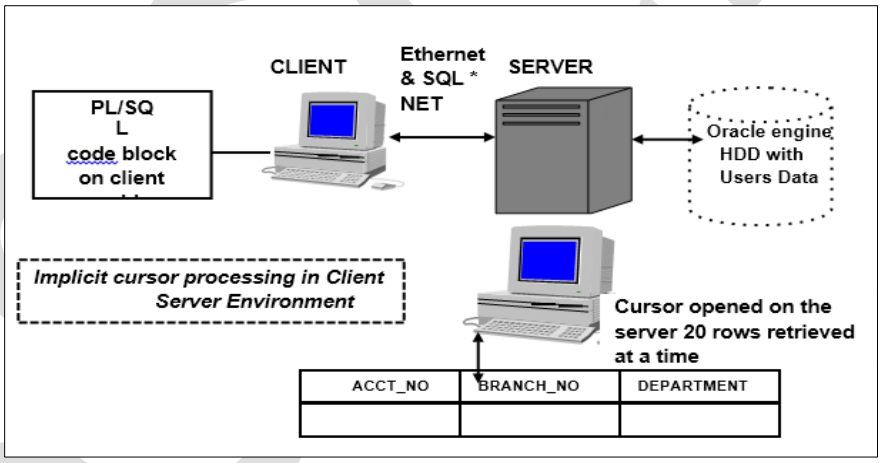
- A cursor is a temporary work area created in the system memory when a SQL statement is executed.
- A cursor contains information of a select statement and the rows of data accessed by it.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- A cursor can hold multiple rows, but can process only one row at a time. The set of rows the cursor holds is called the **active data set**.
- The size of cursor is the sizes require for holding the number of Active data set. There are two types of cursors:
    - **Implicit cursor**
    - **Explicit cursor**

- **Implicit Cursor:**
- Implicit cursor is created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.
- Implicit cursor is automatically open by Oracle Engine. There for we not require to open implicit cursor.
- They are also created when a SELECT statement that returns just one row is executed.
- Oracle provides few attributes known 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.

- 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:**

Attribute Name	Description
%ISOPEN	Returns TRUE if cursor is open, otherwise FALSE
%FOUND	Returns TRUE if record was fetched successfully, otherwise FALSE
%NOTFOUND	Returns TRUE if record was Not Fetched successfully, otherwise FALSE.
%ROWCOUNT	Returns number of records processed from the cursor.

- Since it is opened and managed by the Oracle engine internally, it fulfills the following tasks…
  - Reserving an area in memory.
  - Manipulating this area with appropriate data.
  - Processing the data in the memory area.
  - Releasing the memory area when the processing is complete etc.
  - Result data then passed to the client machine via the network.
- A cursor then opened in memory on the client machine to hold the rows returned by the oracle engine.
- Implicit cursor attributes can be used to access information about the status of the last insert, update, delete or single-row select statements.

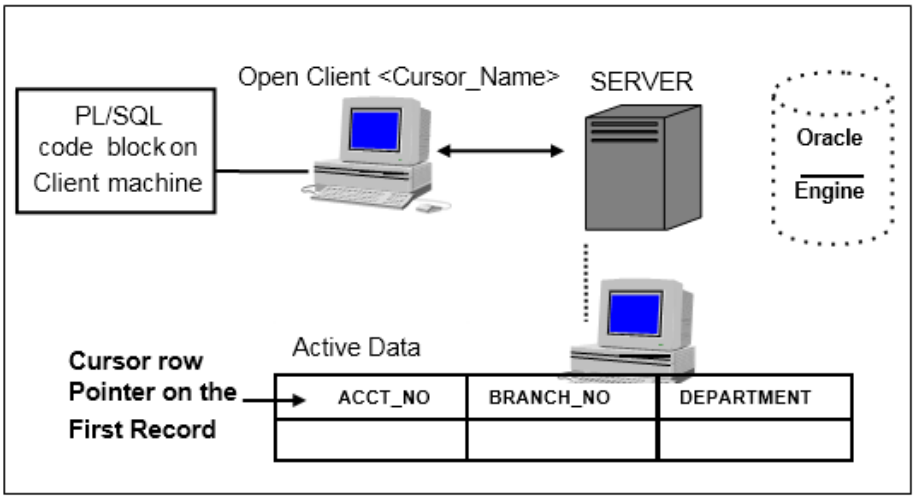


:: Example ::

```
set serveroutput on
set verify off
set feedback off
begin
        update emp set sal=sal+500 where job='MANAGER';
if (sql%found) then
     dbms_output.put_line('Total '||sql%rowcount||' Records are updated');
else
     dbms_output.put_line('No records are Updated');
end if;
end;
```

- **Explicit Cursor Programming**
- Explicit cursor is needed when select statement fetch multiple records from tables.
- It has to be declared manually in the declaration section of the PL/SQL Block.
- Explicit cursor stores multiple records but 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.
- Explicit cursor follows following steps…..
    1. **Declare a cursor**
    2. **Open the cursor**
    3. **Fetch data from the cursor one row at a time into memory variables.**
    4. **Process the data held in the memory variable as required using a loop.**
    5. **Exit from the loop after processing is complete.**
    6. **Close the cursor.**
- **1) Cursor Declaration (Declare a Cursor):**
- Cursor is defined in the declarative part of a PL/SQL block.
- In this section we give the name of cursor.
- Declaration is only virtual it not occupy any memory.
- After declaration three commands used to control the cursor are open, fetch and close.
- **Syntax**: cursor <cursor_name> is <select statement>;
- **Example**: cursor emp_update is select empno, deptno, sal from emp;

- **2) Opening a cursor:**
- It first executes the query and creates the active set that contains all rows which meets the query search criteria.
- When a cursor is opened, the first row becomes the current row.
- Open statement retrieves records from a database table and places the records in the cursor.
- A cursor is opened in the Server's memory.
- **Syntax**: open <cursor_name>;
- **Example**: open emp_update;



- **3) Fetching a record from the Cursor:**
- Fetch statement retrieves the rows from the active set opened in the server into memory variables declared in the PL/SQL code block on the client one row at a time.
- Memory variables are opened at client machine.
- Each time FETCH is executed, the cursor pointer is advanced to the next row in Active Data Set.
- Standard loop structure (loop…End loop) is used to fetch records from the cursor into memory variables one row at a time.
- **Syntax**: FETCH <cursor_name> into <variable1>,<variable2>,….<variableN>;
- **Example**: fetch emp_update into eno, dno, salary;

- **4) Closing a Cursor:**
- Close statement disables the cursor and the active set becomes undefined.
- This will release the memory occupied by the cursor and its data set both on the client and server.
- **Syntax**: close <cursor_name>;
- **Example**: close emp_update;
- **Attributes of Explicit cursor:**

Attribute Name	Description
Cur_name%ISOPEN	Result is TRUE, if an explicit cursor is open else FALSE. To access this attribute.
Cur_name%FOUND	Result is TRUE, if the last fetch succeeded because a row was available or to FALSE if the last fetch failed because no more rows were available.
Cur_name%NOTFOUND	It is logical opposite of %FOUND. It is TRUE, if the last fetch has failed because no more rows were available or to FALSE,
Cur_name%ROWCOUNT	Returns the number of rows fetched from the active set. It is set to zero when the cursor is opened.

- **State the Difference between Implicit & Explicit cursor:**

S_No.	Implicit Cursor	Explicit Cursor
1	It is opened by the oracle engine for its internal processing.	It is declared & opened by the user for data Processing.
2	Default name of the cursor is SQL.	User can create explicit cursor with any name & also change the name.
3	Use SQL prefix before using attributes of implicit cursor. Such as… SQL%FOUND	Use cursor name which is given by the user at declaration time before using attributes of explicit cursor. Such as….cursor_name%FOUND
4	User has no required opening & closing the implicit cursor.	User must have to open the cursor for fetching the data & close the cursor after completion of operation.
5	It has not any type.	It has two types. Normal cursors. Parameterized cursors.
6	We cannot use the FETCH command with implicit cursor.	We must have to use the FETCH command.
7	Query which returns single row implicit cursor are used.	Query which returns multiple rows explicit cursor are used.

- **Parameterize Cursor and Cursor For Loop**
- **Parameterized Cursors:**
- A cursor can be declared with parameters, which allows you to pass values to the cursor.
- Values are passed to the cursor when it is opened, and they are used in the query when it is executed.
- With the use of parameters we can open & close the cursor many times with different values.
- Cursor with different values returns different active data set each time.
- Opening a parameterized cursor & passing values to the cursor open cursor_name (value / variable / Expression);
- **Syntax**: cursor <cursor_name> (Parameter_name data_type) is <select statement>; **Example:**

```
declare
   dno emp.deptno%type;
   cursor cur_dept(no number) is select ename, dname, loc,sal from dept, emp where
   emp.deptno=dept.deptno and dept.deptno=dno;
begin
                dno:=&dno;
                dbms_output.put_line('Employee Name' || 'Department Name' || 'Location' ||
                'salary');
                for cur_record in cur_dept(dno)
                loop
                  dbms_output.put_line(cur_record.ename ||' ' ||cur_record.dname || ' ' ||
                  cur_record.loc || ' '||cur_record.sal);
                end loop;
end;
```

- **Cursor For loops:**
- It is used to process multiple records.
- Advantage of cursor for loop is the loop itself will open a cursor, fetches the records into the cursor from the table until EOF is encounters then it closes the cursor.
- Variable used in cursor for loop is automatically created by the cursor for loop.
- A cursor can be closed even when an exit or a jumping statement is used to leave the loop in advance, of if an exception is raised inside the loop.
- **Syntax:**
- for <variable> in <cursor_name> loop statements
- end loop; **Example:**

```
declare
          cursor cur_emp is select empno,ename,job,sal from emp;
          v_total number(5) :=0;
begin
          for cur_record in cur_emp
          loop
                dbms_output.put_line(cur_record.empno || ' ' || cur_record.ename || '  '  ||
                cur_record.job || '         ' || cur_recorde.sal);
                v_total:=V_total+cur_recor.sal;
          end loop;
                dbms_output.put_line('total salary is :- ' || v_total);
end;
```