Chapter-6

Feature of SQL*Plus

- ✓ SQL plus accepts ad hoc entry of statements.
- ✓ It accepts SQL input from files.
- ✓ It provides a line editor for modifying SQL statements.
- ✓ It controls environmental settings.
- ✓ It formats query results into basic reports.
- ✓ It accesses local & remote database.

Block structure:

The Declare section:

Code blocks start with a declaration section, in which, memory variables and other oracle objects can be declared, and if required initialized. Once declared they can be used in SQL statements for data manipulation.

The Begin section:

It consists of set of SQL and PL/SQL statements, which describe processes that have to be applied to table data. Actual data manipulation, retrieval looping and branching construct are specified in this section.

The Exception section:

This section deals with handling of error that arise during execution of the data manipulation statements, which make up this PL/SQL code block. Errors can arise due to syntax logic and/or validation rule violation.

The End section:

This marks the end of a PL/SQL code block can be diagrammatically represented as follows:

Declare	Declarations of memory variables, constraints, cursors, etc in PL/SQL
Begin	SQL executable statements PL/SQL executable statements
Exception	SQL or PL/SQL code to handle errors that may arise during the execution of the code block between begin and exception section
End;	

PL/SQL Data types:

Both Pl/SQL and oracle have their foundations in SQL. Most PL/SQL data types are native to oracle's data dictionary. Hence there is a very easy integration of PL/SQL code with the oracle engines.

The default data types that can be declared in PL/SQL are:

- Number: For storing numeric data
- Char: For storing character data
- Date: For storing date and time data
- Boolean: For storing True, False or NULL

Number, char and date data types can have null values.

%Type:

Pl/SQL can use the %type attribute to declare variables based on definitions of columns in a table. %type declares a variable or constant to have same data type as that of a previously define variable or of a column in a table or in a view.

Ex:

Name emp_master.ename%type;
No emp_master.eno%type;

So the data type of name and no will be same as that of ename and eno.

%Row type:

For table only.

NOT NULL:

Not Null causes creation of a variable or a constant that cannot be assigned a null value. If an attempt is made to assign the value NULL to a variable or a constant that has been assigned a not null constraint, oracle senses the exception condition automatically can an internal error is returned.

Variable:

Variable in Pl/SQL blocks are named variable. A variable name must begin with a character and can be followed by a maximum of 29 other characters.

Ex.

Rollno number(5);

Assigning values to variables:

Using the assignment operator: =

Rollno :=10;

Selecting or fetching table data values into variables

Constant:

Declaring a constant is similar to declaring a variable except that the keyword constant must be added to the variable name and a value assigned immediately. Thereafter, no further assignments to the constant are possible, while the constant is within the scope of the PL/SQL block.

RAW/Long RAW:

Raw types are used to store binary data. The maximum length of raw variable is 32767 bytes; the maximum length of a database raw column is 255 bytes. These are similar to char variables except that they are not

converting between character sets. It is used to store fixed length binary data.

Long Raw is similar to long data, except that PL/SQL will not convert between character sets. The maximum length of a long raw variable is 32760 bytes. The maximum length of a long raw column is 2 GB.

LOB Type:

The LOB types are used to store large objects. A large object can be either a binary or a character value up to 4 GB in size.

There are four types of LOB's:

BLOB (Binary Large Object):

This store unstructured binary data up to 4 GB in length. A BLOB could contain video or picture information.

CLOB (Character Large Object):

This stores single byte characters up to 4 GB in length. This might be used to store documents.

BFILE (Binary File):

This stores a pointer to read only binary data stored as an external file outside the database. And stores binary pictures.

Displaying user messages on the Screen:

To display messages the SERVEROUTPUT should be set to ON. SERVEROUTPUT is a SQL*PLUS environment parameter that displays the information passed as a parameter to the PUT_LINE function.

Syntax: SERVEROUTPUT OFF/ON;

Ex: set serveroutput on;

COMMENT:

A comment can have two forms as:

The comment line begins with a double hyphen (--). The either line will be treated as a comment.

The comment line begins with a slash followed by an asterisk (/*) till the occurrence of an asterisk followed by a slash (*/). All lines within are treated as comments. This form of specifying comments can be used to span across multiple lines.

CONTROL STRUCTURE:

Conditional Control:

PL/SQL allows the use of an IF statement to control the execution of a block of code. In PL/SQL the (IF THEN ELSE END IF) construct in code block allow specifying certain conditions, under which a specific code block should be executed.

Syntax:

IF <condition> THEN

```
        <Action>
    ELSIF <condition> THEN
        <Action>
    END IF;
```

Example [1]:

```
    Declare
        A Number;
        B Number;
    BEGIN
        A: =10; B: =20;OR A: =&A; B: =&B;
        IF A > B THEN
            DBMS_OUTPUT.PUT_LINE ('A IS BIG' || A);
        ELSE
            DBMS_OUTPUT.PUT_LINE ('B IS BIG' || B);
        END IF;
    END;
```

Example [2]:

```
    Declare
        Commission emp.comm %type
    BEGIN
        SELECT comm INTO commission FROM emp WHERE comm > 100;
        IF commission >=1000 then
            UPDATE emp SET SAL=5000;
        ELSE
            UPDATE emp  SET SAL=4500;
        END IF;
    END;
```

Iterative control:

Iterative control indicates the ability to repeat or skip sections of block. A loop marks a sequence of statements that has to be repeated. While the keyword end loop is placed immediately after the last statement in the sequence. Once a loop being to execute, it will go on forever. Hence a conditional statement in the controls the number of times a loop is executed always accompanies loops.

SIMPLE LOOP:

Syntax: Loop

```
        <Sequence of statement>
     End loop;
```

Example:

```
    Declare
     Number: =0;
    BEGIN
```

```
        LOOP
            i:=i+2;
            EXIT WHEN i>10;
        END LOOP;
    Dbms_output.put_line('Loop exited as the value of I has reached' || to_char(i));
    End;
```

WHILE LOOP:

Syntax: WHILE <Condition>

```
    LOOP
        <Action>
    END LOOP;
```

Example [1]:

```
    CREATE TABLE AREAS (RADIUS number(5),AREA number(14,2));
    Declare
        Pi constraint number (4,2):=3.14;
        Radius number (5); Area number (14,2);
    BEGIN
        Radius :=3;
            WHILE Radius<=7
            LOOP
                Area := pi * power(radius,2);
                INSERT INTO areas VALUES(radius, area);
            END LOOP;
    END;
```

Example [2]: (Program to print fibonacci series)

```
    Declare
        X number (5);Y number(5);i number(4);
    BEGIN
        x:=1;y:=0;i:=0;
        While X<=100
        LOOP
            i:=Y;
            Y:=X;
            X:=i+Y;
            Dbms_output.put_line (to_char(Y));
        End loop;
END;
```

The FOR LOOP:

Syntax:

```
    FOR variable IN [REVERSE] Start .. End
    LOOP
        <Action>
```

```
        END LOOP;
```

Example [1]:

```
        Declare
                Given_number varchar2(5):='1234';
                Str_length number(2);
                Inverted_number varchar2(5);
        BEGIN
                Str_length:=length(given_number);
                FOR cntr IN REVERSE 1 .. str_length
                LOOP
                        Inverted_number:=Inverted_number ||
substr(Given_number,cntr,1);
                END LOOP;
                Dbms_output.put_line ('The Given Number is' || Given_number);
                Dbms_output.put_line ('The inverted Number is' || Inverted_number);
        END;
```

Example [2]:

```
        Declare
                        A number: =0;

        Begin

                For I in 0..9

                Loop

                        Dbms_output.put_line ('Result of I is :' || to_char(i));

                end loop;

                Dbms_output.put_line ('In Reverse');

                For I in reverse 0..9

                Loop

        Dbms_output.put_line ('Result of I is :' || to_char(i));End loop;
        End;
```

The GOTO statement

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags <<user-defined name>. The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

Syntax: GOTO <codeblock name>;

Example:

```
Declare
    Balance number(10);
Begin
    Select sell_price into balance from product_master
    where pro_no = 'p001';
    if balance < 4000 then
        goto add_price;

    else
        dbms_output.put_line('current price of product is :'|| balance);
    end if;
<<add_price>>
    update product_master set sell_price = 4000 where pro_no = 'p001';
    insert into old_price_table (pro_no, date_change, old_price)
    values('p001',sysdate,balance);
    dbms_output.put_line('the value of product is 4000');
End;
```

SAVEPOINT:

Main syntax of rollback is

Rollback [work] [to [savepoint] savepoint name];

Here savepoint marks and saves the current point in the processing of a transaction. When a savepoint is used with a rollback statement, parts of a transaction can be undone.

Syntax:

Savepoint savepoint name;

Example:

```
Declare
    Total_sal number(10);
Begin
    Insert into emp values('e006','jhon',1000);
    Savepoint no_update;
    Update emp set sal = sal +2000 where name = 'abc';
    Update emp set sal = sal +2000 where name = 'xyz';

    Select sum(sal) into total_sal from emp;
    If total_sal > 20000 then
        Rollback to savepoint no_update;
    End if;
```

```
        Commit;
End;
```

CURSOR:

What is cursor ?

The oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL's operations and is called a cursor. The data that is toed in the cursor is called the active data set. Conceptually the size of the cursor in memory is the size required to hold the number of rows in the active data set.

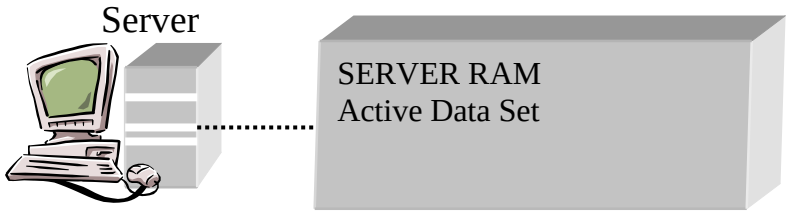
The oracle engines built in memory management capabilities and the amount of RAM available however determine the actual size. Oracle has a pre-defined area in main memory set aside, within which cursors are opened. Hence the cursor's size will be limited by the size of this pre-defined area. The values retrieved from a table are held in a cursor opened in memory by the oracle engine. This data is then transferred to the client machine via the network.

In order to hold this data a cursor is opened at the cline end. Id the number of rows returned by the oracle engine is more than the area available in the cursor opened on the client, the cursor data and the retrieved data is swapped between the operation system' swap area and RAM.

When a user fires a select statement as:

```
SELECT emp_no, name, dept from emp_mstr where branch_no='B1';
```

The resultant data set in the cursor opened at the server end is displayed as under:



When a cursor is loaded with multiple rows via a query the oracle engine opens and maintains a row pointer. Depending on user's requests to view data the row pointer will be relocated within the cursor's active data set. Additionally oracle also maintains mult8iple cursor variables. The values held in these variables indicated the status of the processing being done by the cursor.

Types of Cursors:

Cursors are classified by depending on the circumstances under which they area opened. Id the oracle engine opened a cursor for its internal processing it is known as **Implicit cursor**. A cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an **explicit cursor**.

General Cursor Attributes:

When the oracle engine creates an Implicit or Explicit cursor, cursor control variables are also created to control the execution of the cursor.

Attributes	Description
%ISOPEN	Returns True if cursor is open, False otherwise
%FOUND	Returns True if record was fetched successfully, False otherwise
%NOTFOUND	Returns True if record was not fetched successfully, False otherwise
%ROWCOUNT	Returns number of records processed from the cursor

Implicit cursor:

The oracle engine implicit opens cursor on the server to process each SQL statement. Since the implicit cursor is opened and managed by the oracle engine internally. The function of reserving an area in memory, populating this area with appropriate data, processing the data in the memory area, releasing the memory area when the processing is complete is taken care of by he oracle engine.

The resultant data is then passed to the client machine vial the network. A cursor is then opened in memory on the client machine to hold the row returned by the oracle engine. The client's operating system and its swap area manage the number of rows held in the cursor on the client

Implicit cursor attributes can be used to access information about the status of the last insert, update, delete or single row select statements. Preceding the implicit cursor attribute with the cursor name can do this. The values of the cursor attributes always refer to the most recently executed SQL statement whenever the statement appears. If an attribute value is to be saved for later use, it must be assigned to a memory variable.

Example [1]:

```
Declare
        Emp_no1 varchar2 (5);
        Emp_nm varchar2(10);
BEGIN
        Emp_no1 = &Emp_no1;
        Emp_nm = &Emp_nm;
        UPDATE employee
        SET emp_no=emp_no1 WHERE emp_name=&emp_nm;
        IF SQL%FOUND then
                Dbms_output.put_line ('Employee transferred');
        END IF;

        IF SQL%NOTFOUND then
```

```
            Dbms_output.put_line ('Employee not transferred');
        End IF;
End;
```

Example [2]:

```
Declare
        V_count  char(4);
Begin
        UPDATE employee
        SET salary = salary + (salary * 0.5)
        WHERE job = 'Computer';
        V_count := to_char(SQL%ROWCOUNT);
        IF SQL%ROWCOUNT>0 THEN
            Dbms_output.put_line(v_count || 'Records are updated');
        ELSE
            Dbms_output.put_line('Record not updated');
        END IF;
END;
```

Explicit Cursor:

When individual records in a table have to be processed inside a PL/SQL code block a cursor is used. This cursor will be declared and mapped to an SQL query in the Declare Section f the PL/SQL block and used within its Executable Section. A cursor thus created and used is known as an Explicit Cursor.

Explicit Cursor Management:

- Declare a cursor mapped to a SQL select statement that retrieves data for processing.
- Open the cursor.
- Fetch data from the cursor one row at a time into memory variables.
- Process the data held in the memory variables as required using a loop.
- Exit from the loop after processing is complete.
- Close the cursor.

The Functionality of Open, Fetch and Close Command:

- Defines a private SQK area named after the cursor name.
- Executes a query associated with the cursor.
- Retrieves table data and populate the named SQL area in memory i.e. creates the active data set.
- Sets the cursor row pointer in the active data set to the first record.

A fetch statement is placed inside a Loop…End Loop construct, which causes the data to be fetched into the memory variables and processed until all the rows in the rows in the Active Data Set are processed. The fetched into loop then exits. The exiting of the fetch loop is user controlled.

After the fetch loop exits, the cursor must be closed with the close statement. This will release the memory occupied by the cursor and its Active Data Set. The cursor name is used to reference the Active Data Set held within the cursor.

Declaring a Cursor:

CURSOR <cursor Name> IS SQL statement;

Opening A Cursor

Syntax: OPEN Cursor Name;

Fetching A Record From The Cursor

Syntax: FETCH Cursor Name INTI Variable1, Variabke2…;

Closing A Cursor

Syntax: CLOSE Cursor Name;

Example [1]:

```
Declare
    Cursor c_emp is
        Select emp_no, salary from employee Where deptno='d01';
    emp_code employee.emp_no%type;
    sal employee.salary%type ;
Begin
    Open c_emp;
    Loop
        Fatch c_emp into emp_code, sal;
        Exit When c_emp%notfound;
        UPDATE employee SET salary=sal + (sal * 0.05)
        WHERE emp_no = emp_code;
    End loop;
    Commit;
    Close c_emp;
End;
```

Example [2]:

```
Declare
    Cursor cur_count is select name,dept.dept_no,salary from
    employee,dept where employee.dept_no = dept.dept_no;

    ename employee.name%type;
    dno employee.deptno%type;
    sal employee.salary%type;
BEGIN
    OPEN cur_count;
    Dbms_output.put_line('Name      Department      Salary');
    Dbms_output.put_line('--------  --------------  --------');
```

```
        LOOP
            FATCH cur_count INTO ename, dno, sal;
            Dbms_output.put_line(ename || '   '||dno || ' '|| sal);
            EXIT when cur_count%ROWCOUNT>3;
        END LOOP;
END;
```

****