SSN COLLEGE OF ENGINEERING – KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject Code: UCS1404 **Subject:** DBMS
Name of the staff: Mr. N. Sujaudeen **Class:** B.E. - IV Sem

Database (Back-end) : Oracle 10g

PL/SQL – CONTROL STRUCTURES & CURSORS

PL/SQL is a block-structured language. That is, the basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which contain any number of nested sub-blocks. PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

PL/SQL lets you use all the SQL data manipulation, cursor control, and transaction control commands, as well as the SQL functions, operators. PL/SQL allows the applications to be written in a PL/SQL procedure or a package and stored at Oracle server, where these PL/SQL codes can be used as shared libraries, or applications, thus enhancing the integration and code reuse.

PL/SQL has three parts: a declarative, an executable part, and an exception-handling part. Only the executable part is compulsory. Items can be declared in declarative part. Once declared, items can be manipulated in the executable part. Exceptions raised during execution can be dealt with in the exception-handling part.

Block Structure

```
[ DECLARE
     --  declarations ]
 BEGIN
     --  statements
[ EXCEPTION
     --  handlers ]
END;
```

For example:

```
declare
temp number(3);
begin
      temp:=3;
      dbms_ouput.put_line( 'Value of temp is =  '  ||  temp );
end;
```

Note that **dbms_output** is an Oracle-supplied PL/SQL package and **put_line** is one of the packaged procedures. It displays the values on the SQL*Plus terminal which must be enabled with SET SERVEROUTPUT ON command.

Advantages of PL/SQL

PL/SQL is a completely portable, hight-performance transaction processing language that offers the following advantages:

- Support for SQL
- Support for object-oriented programming
- Better performance
- Higher productivity
- Full portability
- Tight integration with Oracle
- Tight security

&&&&&&&&&&&&&&&&&&&&&&&&&&

PL/SQL CONTROL STRUCTURES

Conditional Control

IF Statement: Syntax

```
IF condition THEN
      sequence_of_statements
END IF;
```

```
IF condition THEN
      sequence_of_statements1
ELSE
      sequence_of_statements2
END IF;
```

```
IF condition1 THEN
     sequence_of_statements1
ELSIF condition2 THEN
     sequence_of_statements2
ELSE
     sequence_of_statements3
END IF;
```

Case Statement: Syntax

```
CASE selector
     WHEN expression1 THEN sequence_of_statements1;
     WHEN expression1 THEN sequence_of_statements2;
     .......
     WHEN expression1 THEN sequence_of_statementsN;
     [ELSE sequence_of_statementsN+1]
END CASE;
```

Iterative Control:

a) Simple Loop: Syntax

```
LOOP
     sequence_of_statements
     EXIT WHEN condition;
END LOOP;
```

Use EXIT which forces a loop to complete unconditionally or use EXIT-WHEN statement that lets a loop complete conditionally.

b) While Loop: Syntax

```
WHILE condition LOOP
     sequence_of_statements
END LOOP;
```

c) For Loop: Syntax

```
FOR counter IN [REVERSE]  lower_bound . . higher_bound  LOOP
     sequence_of_statements
END LOOP;
```

PL/SQL Attributes

PL/SQL variables and cursors have *attributes*, which are properties that let you reference the datatype and structure of an item without repeating its definition. A percent sign (%) serves as the attribute indicator.

a) %TYPE

The %TYPE attribute provides the data type of a variable or database column. This is particularly useful when declaring variables that will hold database values. In the following example, %TYPE provides the data type of a variable:

```
credit REAL(7,2);
debit credit%TYPE;
```

The %TYPE attribute is particularly useful when declaring variables that refer to database columns. For example: consider a column named *title* in table named *books*.

```
my_title books.title%TYPE
```

Now the variable my_title has the same datatype as column *title* in *books* relation.

%ROWTYPE

The %ROWTYPE attribute provides a record type that represents a row in a table (or view). The record can store an entire row of data selected from the table or fetched from a cursor.

```
DECLARE
    emp_rec   emp%ROWTYPE          -- stores a row selected from the emp table
    CURSOR c1 IS SELECT deptno, dname, loc FROM dept;
    dept_cur  c1%ROWTYPE           -- stores a row fetched from cursor c1
```

Columns in a row and corresponding fields in a record have the same names and datatypes. To reference a field use dot notation.

CURSORS

Oracle uses work areas to execute SQL statements and store processing information. A PL/SQL construct called a *cursor* lets you name a work area and access its stored information. There are two kinds of cursors: *implicit* and *explicit.* PL/SQL declares a cursor implicitly for all SQL data manipulation statements, including queries that return only one row. For a queries that return multiple rows, you can explicitly declare a cursor to process the rows individually. An example follows:

```
DECLARE
      CURSOR c1 IS
            SELECT empno, ename, job FROM emp WHERE  deptno = 20;
```

The set of rows returned by a multi-row query is called the *result set*. Its size is the number of rows that meet your search criteria.

Declaring an Explicit Cursor:

```
      CURSOR  cursor_name [ ( parameter [, parameter].....) ]
      [ RETURN return_type ]  IS  select_statement;
```

Where return_type must represent a record or a row in a database table, and parameter stands for the following syntax:

cursor_parameter_name [IN] datatype [{ := | DEFAULT } expression]

IMPLICIT CURSORS

Oracle implicitly opens cursor to process each SQL statement not associated with an explicit cursor. PL/SQL lets you refer to the most recent implicit cursor as the SQL cursor, which always has these attributes: %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. These attributes were always used along with SQL. Ex: SQL%FOUND, SQL%ROWCOUNT

1) %FOUND – This attribute yields TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it yields FALSE.
2) %ISOPEN – This attribute always yields FALSE because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
3) %NOTFOUND – This attribute is the logical opposite of %FOUND.
4) %ROWCOUNT – This attribute yields the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

5) SQL – This is the name of the Oracle implicit cursor.

EXPLICIT CURSORS

To execute a multi-row query, Oracle opens an unnamed work area that strores processing information. An explicit cursor lets you name the work area, access the information, and process the rows individually. Moreover, you can declare a cursor in the declarative part of any PL/SQL block, subprogram, or package.

Use three commands to control a cursor: OPEN, FETCH, and CLOSE. First, initialize the cursor with the OPEN statement, which identifies the resultset. Then, execute FETCH repeatedly until all rows rows have been retrieved. When the last row has been processed, release the cursor with the CLOSE statement.

Opening a cursor:

```
DECLARE
     CURSOR c1 IS SELECT enom, ename, d_no FROM emp WHERE sal > 3000;
     .....
BEGIN
     OPEN c1;
     ...
END;
```

Note: Rows in the result set are not retrieved when the OPEN statement is executed. Rather, the FETCH statement retrieves the rows.

Fetching with a Cursor:

The FETCH statement retrieves the rows in the resultset one at a time. After each fetch, the cursor advances to the next row in the resultset. An example:

```
     FETCH  c1 INTO my_empno, my_ename, my_deptno;
```

For each column value returned by the query associated with the cursor, there must be a corresponding, type-compatible variable in the INTO list.

Closing a Cursor:

The CLOSE statement disables the cursor, and result set becomes undefined. Once the cursor is

closed, you can reopen it. Any other operation on a closed cursor raises the exception.

Cursor FOR Loops:

In most situations that require an explicit cursor, you can simplify coding by using a cursor FOR loop instead of the OPEN, FETCH, and CLOSE statements. A cursor FOR loop implicitly declares its loop index as a record that represents a row fetched from the database. Next, it opens a cursor, repeatedly fetches rows of values from the result set into the fields in the record, then closes the cursor when all rows have been processed.

&&&&&&&&&&&&&&&&&&