Database Systems
Lab10

PL/SQL
Control Statements

Input in SQL

```
DECLARE
        V_NUM NUMBER(2);
BEGIN
        V_NUM := &V_NUM;
        DBMS_OUTPUT.PUT_LINE (V_NUM * 10);
END;
```

PL/SQL Selection Structures (IF Statement)

- **IF/END IF:**

  IF *condition* THEN
  program statements
  END IF;

- **IF/ELSE/END IF:**

  IF *condition* THEN
  program statements
  ELSE
  alternate program statements
  END IF;

PL/SQL Selection Structures (IF Statement)

- **IF/ELSIF:**

```
IF condition1 THEN
        program statements;
ELSIF condition2 THEN
        alternate program statements;
ELSIF condition3 THEN
        alternate program statements;
. . .
ELSE
        alternate program statements;
END IF;
```

PL/SQL Comparison Operators

Operator	Description	Example
=	Equal	Count = 5
<>, !=	Not Equal	Count <> 5
>	Greater Than	Count > 5
<	Less Than	Count < 5
>=	Greater Than or Equal To	Count >= 5
<=	Less Than or Equal To	Count <= 5

Evaluating NULL Conditions in IF/THEN Structures

- **If a condition evaluates as NULL, then it is FALSE**

- **How can a condition evaluate as NULL?**
  - **It uses a BOOLEAN variable that has not been initialized**
  - **It uses any other variable that has not been initialized**

Example:

```
IF acct_balance >= debit_amt THEN
    UPDATE accounts SET bal = bal - debit_amt
      WHERE account_id = acct;
ELSE
    INSERT INTO temp
    VALUES (acct, acct_balance, 'Insufficient funds');
END IF;
```

PL/SQL Loops

- Loop: repeats one or more program statements multiple times until an exit condition is reached
  - Pretest loop: exit condition is tested before program statements are executed
  - Posttest loop: exit condition is tested after program statements are executed

LOOP … EXIT Loop

```
LOOP
   program statements
   IF condition THEN
     EXIT;
   END IF;
   more program statements
END LOOP;
```

Pretest
OR
Posttest

LOOP … EXIT WHEN Loop

```
LOOP
   program statements
   EXIT WHEN condition;
END LOOP;
```

Posttest

WHILE Loop

```
WHILE condition
LOOP
   program statements
END LOOP;
```

Pretest

Numeric FOR Loop

```
FOR counter_variable
IN start_value .. end_value
LOOP
   program statements
END LOOP;
```

Preset number of iterations

Example:

```
FOR num IN 1..500 LOOP
   INSERT INTO roots VALUES (num, SQRT(num));
END LOOP;
```

Example:

```
WHILE salary <= 2500 LOOP
    SELECT salary, mgr_ssn, lname
    INTO   salary, mgr_ssn, last_name
    FROM   employee
    WHERE  ssn = mgr_ssn;
END LOOP;
```

Cursors

- A cursor is a pointer to a private SQL area that stores results of a SELECT statement.

- **Types of Cursors**
  - Implicit
  - Explicit

Cursors

- **Implicit Cursors**
  - Created automatically every time you use an INSERT, UPDATE, DELETE, or SELECT command
  - Doesn't need to be declared
  - Can be used to assign the output of a SELECT command to one or more PL/SQL variables
  - Can only be used if query returns one and only one record

Cursors

- Must be declared in program DECLARE section
- Can be used to assign the output of a SELECT command to one or more PL/SQL variables
- Can be used if query returns multiple records or no records

Using an Explicit Cursor

- Declare the cursor
- Open the cursor
- Fetch the cursor result into PL/SQL program variables
- Close the cursor

Declaring an Explicit Cursor

DECLARE

CURSOR *cursor_name* IS *SELECT_statement*;

BEGIN

-- Program statements

END;

Opening an Explicit Cursor

OPEN *cursor_name*;

Fetching Explicit Cursor Records

FETCH *cursor_name*
INTO *variable_name(s)*;

Closing an Explicit Cursor

CLOSE *cursor_name*;

Processing an Explicit Cursor

- **LOOP ..EXIT WHEN approach:**

OPEN *cursor_name*;
LOOP
    FETCH *cursor_name* INTO v*ariable_name(s)*;
    EXIT WHEN *cursor_name*%NOTFOUND:
END LOOP;
CLOSE *cursor_name*;

Processing an Explicit Cursor

- **Cursor FOR Loop approach:**

FOR *variable_name(s)* in *cursor_name* LOOP
    additional processing statements;
END LOOP;

Explicit Cursor Attributes

Attribute	Return Value
%NOTFOUND	TRUE when no rows left to fetch; FALSE when rows left to fetch
%FOUND	TRUE when rows left to fetch; FALSE when no rows left to fetch
%ROWCOUNT	Number of rows a cursor has fetched so far
%ISOPEN	TRUE if cursor is open and FALSE is cursor is closed

Using Reference Data Types in Explicit Cursor Processing

- Declaring a ROWTYPE reference variable:

DECLARE
reference_variable_name cursor_name%ROWTYPE;

- Referencing a ROWTYPE reference variable:

reference_variable_name.database_field_name

Example1:

```
DECLARE
    Emp_name    VARCHAR2(10);
    Cursor      c1 IS SELECT Ename FROM Emp_tab
                        WHERE Deptno = 20;
BEGIN
  OPEN c1;
     LOOP
        FETCH c1 INTO Emp_name;
        EXIT WHEN c1%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(Emp_name);
    END LOOP;
END;
```

Example2:

```
DECLARE
  CURSOR c1 is
    SELECT fname, ssn, salary FROM employee
      ORDER BY salary DESC; --start w/ highest paid emp
  my_ename VARCHAR2(10);
  my_empno CHAR(9);
  my_sal   NUMBER(10,2);
BEGIN
  OPEN c1;
  FETCH c1 INTO my_ename, my_empno, my_sal;
  WHILE C1%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE (MY_EMPNO||','||MY_SAL);
    UPDATE employee
    SET salary = salary * 1.1
     WHERE ssn = my_empno;
/*By this statement you will update only the employees retrieved by the cursor.*/
    FETCH c1 INTO my_ename, my_empno, my_sal;
  END LOOP;
CLOSE c1;
COMMIT;    /* This will save all updates applied on Employee table */
EXCEPTION
   WHEN OTHERS THEN
       ROLLBACK;
/*If any error occurred, any updates applied on the employee table will not be
saved and reversed as the state it was before applying this program */
END;
```