DATABASE MANGEMENT SYSTEMS

B.E V SEM ‘A’ SEC

P.Mirunalini

AP ,CSE

SSNCE

PL/SQL – CONTROL STRUCTURES & CURSORS

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

PL/SQL – CONTROL STRUCTURES & CURSORS

- PL/SQL 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,
- These PL/SQL codes can be used as shared libraries, or applications, thus enhancing the integration and code reuse.

PL/SQL Parts

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

PL/SQL Syntax

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

Advantages of PL/SQL

- PL/SQL is a completely portable, high-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

- IF condition THEN
  - sequence_of_statements
- END IF;
- IF condition THEN
  - sequence_of_statements1
- ELSE
  - sequence_of_statements2
- 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

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

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 has the properties of referring the datatype and structure of an item without repeating its definition.
- A percent sign (%) serves as the attribute indicator.
- **%TYPE**
  - The %TYPE attribute provides the **datatype of a variable or database column.**
  - This is particularly useful when declaring variables that will hold database values.
    - **credit REAL(7,2);**
    - **debit credit%TYPE;**
  - The %TYPE attribute is particularly useful when declaring variables that refer to database columns.

my_title books.title% TYPE

PL/SQL Attributes-Example

--table ctreation

Create table T2(a number(3), b char(1));

---creating procedure

```
CREATE or Replace PROCEDURE addtuple2(
   x T2.a%TYPE,
    y T2.b%TYPE)
AS
BEGIN
      INSERT INTO T2 VALUES (x, y);
END;
```

---calling procedure

```
Call addtuple2(10,'p');
```

PL/SQL Attributes

- **%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)

PL/SQL Attributes

- %ROWTYPE with columns from multiple tables.

```
CURSOR c2 IS
    SELECT employee_id, email, employees.manager_id,
    location_id
    FROM employees, departments
    WHERE
    employees.department_id = departments.department_id;
join_rec c2%ROWTYPE;
```

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

PL/SQL Attributes-Example

```
Declare
    e t2%rowtype;
    begin
        select * into e from t2;
         if e.a=10 then
                update t2 set b='o';
        end if;
    end;
```

CURSORS

- Oracle uses **work areas** to **execute SQL statements** and **store processing information.**
- A PL/SQL construct called a ***cursor*** 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.

CURSORS

DECLARE

CURSOR c1 IS

SELECT empno, ename, job FROM emp WHERE deptno = 20;

- The set of rows returned by a multirow 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.

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**
  - **%ROWCOUNT**.
- These attributes were always used along with SQL.
- Ex: SQL%FOUND, SQL%ROWCOUNT

IMPLICIT CURSORS

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.

IMPLICIT CURSORS-EXAMPLE

---TABLE CREATION

CREATE TABLE dept_temp AS SELECT * FROM departments;

-----IMPLICT CURSOR IMPLEMENTATION

```
DECLARE
 dept_no NUMBER(4) :=190;
      BEGIN
           DELETE FROM dept_temp WHERE department_id = dept_no;
            IF SQL%FOUND THEN -- delete succeeded
           INSERT INTO dept_temp VALUES (270, 'Personnel', 200, 1700);
          END IF;
      END;
```

EXPLICIT CURSORS

- To execute a multirow query, Oracle opens an unnamed work area that stores processing information.
- For an explicit cursor give name for work area, access the information, and process the rows individually.
- 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**.
- **Initialize the cursor** with the **OPEN** statement, which identifies the resultset.
- **Execute FETCH** repeatedly until **all rows** have been **retrieved.**
- When the **last row has been processed**, **Release** the cursor with the **CLOSE statement.**

EXPLICIT CURSORS

```
DECLARE
    CURSOR c1 IS SELECT ename, job,dept_no FROM emp
    WHERE sal > 3000;
    BEGIN
        OPEN c1;
        ……..
    END;
```

- 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.

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

- An explicit cursor, simplifies 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
- Closes the cursor when all rows have been processed.

Explicit cursor-examples

```
DECLARE
    CURSOR c1 IS SELECT last_name, salary FROM employees WHERE
    ROWNUM < 11;
    my_ename employees.last_name%TYPE;
    my_salary employees.salary%TYPE;
    BEGIN
        OPEN c1;
        LOOP
                FETCH c1 INTO my_ename, my_salary;
                IF c1%FOUND THEN -- fetch succeeded
                        DBMS_OUTPUT.PUT_LINE('Name = ' || my_ename ||
                        ', salary = ' || my_salary);
                ELSE -- fetch failed,
                        EXIT;
                END IF;
        END LOOP;
    END;
```