Oracle PL/SQL III

Cursors

Procedures

Functions

iSQLplus: http://uadisq01.uad.ac.uk:5560/isqlplus

Remember the SELECT INTO…… ?

- It only allowed the retrieval of one row

```
Select attribute into variable from … where …

Or

Select count(*) into variable from ………
```

But when we want to retrieve multiple rows we need to use what is called a CURSOR

What is the Cursor structure?

- Declare it
  - This is achieved by a SELECT command
  - And by giving the CURSOR a name
- Open it
- Fetch row(s) from it
- Close it

Declaring the Cursor

```
DECLARE
  CURSOR low_pay
        IS SELECT surname,salary
        FROM Personnel
        where salary < 12000;
v_surname             personnel.surname%TYPE;
v_salary              personnel.salary%TYPE;
BEGIN
…. .
```

Because a cursor is associated with multiple rows they are normally used with LOOP structures

Block1

OPEN, FETCH, CLOSE, %NOTFOUND

```
DECLARE
   CURSOR low_pay IS SELECT surname,salary FROM
   Personnel where salary < 30000;
v_surname                personnel.surname%TYPE;
v_salary                 personnel.salary%TYPE;
BEGIN
   OPEN low_pay;
      LOOP
           FETCH low_pay INTO v_surname, v_salary;
             EXIT when low_pay%NOTFOUND;
             DBMS_OUTPUT.PUT_LINE(v_surname ||' '||
   v_salary);
      END LOOP;
   CLOSE low_pay;
END;
```

A variation using WHILE Loop and %FOUND

```
DECLARE
   CURSOR low_pay IS SELECT surname,salary FROM
   Personnel where salary < 30000;
v_surname                  personnel.surname%TYPE;
v_salary                   personnel.salary%TYPE;
BEGIN
   OPEN low_pay;
    FETCH low_pay INTO v_surname, v_salary;
         WHILE low_pay%FOUND LOOP
           DBMS_OUTPUT.PUT_LINE(v_surname ||' '||
   v_salary);
           FETCH low_pay INTO v_surname, v_salary;
      END LOOP;
   CLOSE low_pay;
END;
```

Note 2 FETCH commands

Block2

Parameters in Cursors

```
DECLARE
   CURSOR c_salary (p_min number,p_max number)
    IS SELECT surname,salary FROM Personnel
         where salary between p_min and p_max;
v_surname           Personnel.surname%TYPE;
v_salary      Personnel.salary%TYPE;
BEGIN
   OPEN c_salary(&p_min, &p_max);
    LOOP
         FETCH c_salary INTO v_surname, v_salary;
         EXIT WHEN c_salary%NOTFOUND;
           DBMS_OUTPUT.PUT_LINE(v_surname||' '||v_salary);
    END LOOP;
   CLOSE c_salary;
END;
```

These would be in quotes for VARCHAR2 variables

FOR LOOP requires no CURSOR OPEN, FETCH, CLOSE

```
DECLARE
   CURSOR c_salary IS SELECT surname,salary
         FROM Personnel
         where salary < 30000;
BEGIN
   FOR counter in c_salary LOOP
         DBMS_OUTPUT.PUT_LINE(counter.surname
                ||' '||counter.salary);
   END LOOP
END;
```

Useful when updating or deleting each row fetched in a cursor otherwise all would be updated at once

SELECT FOR UPDATE Cursors

```
DECLARE
   CURSOR c_salary IS SELECT surname,salary FROM Personnel
         FOR UPDATE;
v_surname           personnel.surname%TYPE;
v_salary      personnel.salary%TYPE;
BEGIN
   OPEN c_salary;
    LOOP
         FETCH c_salary INTO v_surname, v_salary;
         EXIT WHEN c_salary%NOTFOUND;
         UPDATE Personnel SET BONUS=v_salary*0.05 WHERE
               CURRENT of c_salary;
    END LOOP;
   CLOSE c_salary;
END;
```

Stored Procedures

```
CREATE OR REPLACE PROCEDURE proc_name
IS
<declarations of variables, cursors etc> …….. 
BEGIN
  <executing code> ….. 
END proc_name;
```

- A unit of code that performs one or more tasks
- After completion, execution returns to the calling block
- To run the procedure at any time, use **EXECUTE <procedure_name>**

Example Procedure with a cursor

```
CREATE OR REPLACE PROCEDURE surnames
IS
   CURSOR c_staff
   IS
   SELECT surname
   FROM Personnel
   where div=10;
BEGIN
   FOR names IN c_staff LOOP
         DBMS_OUTPUT.PUT_LINE(names.surname);
   END LOOP
END;
```

The message ‘Procedure created’ should be displayed

Block3

Example Procedure to update salaries and how to test it

```
CREATE OR REPLACE PROCEDURE sal_update
IS
BEGIN
  UPDATE personnel set salary=salary*1.1
        where div=10;
END;
```

Execute sal_update;
Select salary from personnel where div=10;
    - to test the procedure

Passing parameters

Parameter name   Parameter mode   datatype

```
CREATE OR REPLACE PROCEDURE test
(firstpar     IN    varchar2,
   secondpar    IN    Date)
IS
   empname      varchar2(30);
   empid number(8);
BEGIN
   ……. .
END;
```

Notice parameter declarations are unconstrained

IN is the default if no mode specified

IN

```
CREATE OR REPLACE PROCEDURE t
( p_salary    IN    Number)
  IS
BEGIN
  ……. .
  p_salary:=p_salary + 100;
END;
```

This is illegal as parameter can only be ***referenced***, not ***changed***

This is legal as parameter is assigned to a variable first

```
CREATE OR REPLACE PROCEDURE t
( p_salary    IN    Number,
  IS
  v_salary    number(15);
BEGIN
  ……. .
  v_salary:=p_salary + 100;
END;
```

Block4

IN Example

```
CREATE OR REPLACE PROCEDURE proc_IN
( p_branchNumber    IN    Number,
  p_percent         IN    Number)
IS
BEGIN
  UPDATE Personnel set
  salary=salary+salary*p_percent/100
  where div = p_branchNumber;
END;
```

EXECUTE proc_IN(10,25)

EXECUTE updperc(10,3455)

Actual and Formal parameters

```
CREATE OR REPLACE PROCEDURE updperc
( p_percent    IN     Number,
  p_Emp IN     Number)
IS
  CURSOR staff_cur IS
    SELECT joindate, div
    from Personnel
    where managedBy=p_Emp;
BEGIN
  For stf in staff_cur LOOP
        updStaff(stf.joindate,stf.div);
  END LOOP;
  …
END;
```

Formal

```
CREATE OR REPLACE
          PROCEDURE
  updstaff
( p_joindate IN Date,
  p_div  IN Number)
IS
….
```

Actual

The Calling Procedure

Block5

Another Calling Example

```
DECLARE
   v_var NUMBER := 20;
BEGIN
   delete_staff(v_var);
END;

CREATE OR REPLACE PROCEDURE delete_staff
(p_branchNumber  IN   Number)
IS
BEGIN
    DELETE Personnel
    WHERE div=p_branchNumber;
END;
```

Anonymous block calls procedure

Using parameters in Loops

```
CREATE OR REPLACE PROCEDURE
insert_root (from_val NUMBER, to_val NUMBER)
IS
num NUMBER;
BEGIN
FOR num IN from_val .. to_val LOOP
  INSERT INTO roots VALUES (num, SQRT(num));
END LOOP;
END;
```

To execute this procedure (e.g insert values from 30 to 32)

```
EXECUTE insert_root(30,32)
```

FUNCTIONS

- Functions are similar to procedures
- They are used for calculations and returning a value

```
CREATE OR REPLACE FUNCTION
   function_name     (parameter
   list)
RETURN return_datatype
IS
   … variables, cursors etc
BEGIN
   Execution code ………………;
   Return expression;
END;
```

Can be:
NUMBER
VARCHAR2
BOOLEAN
 etc

RETURN Statement

- Determines
  - The point at which execution returns to the calling block AND the value that is assigned to it
- RETURN expression
  - Where expression can be any legal PL/SQL expression

```
v_salary := get_salary(10)
```

Block calls the function get_salary for employee 10
Get_salary will return the salary for employee 10
and this will be assigned to v_salary

Block6

Example Function

```
CREATE OR REPLACE FUNCTION
   get_aveSal
   (i_div IN NUMBER)
   RETURN number
IS
v_salary personnel.salary
   %type;
BEGIN
   SELECT avg(salary)
   INTO v_salary FROM
   Personnel
   WHERE div=i_div;
   RETURN v_salary;
END get_aveSal;
```

```
SET SERVEROUTPUT ON
DECLARE
V_divID             personnel.div%type;
v_divName           branch.DivName%type:='&divName';
V_aveSalary    personnel.salary%type;
BEGIN
SELECT div into v_divID
FROM branch WHERE divname=v_divName;
v_aveSalary:=get_aveSal(v_divID);
DBMS_OUTPUT.PUT_LINE('Division '||v_divID||' has '||
    v_aveSalary||' average salary');
END;
```

"get the average salary for ADMIN" Block prompts for division name then passes the division number to the function get_aveSal

Summary

- CURSORS
  - To process all rows of a selected set
- STORED PROCEDURES
  - Parameters
  - Calling them
- FUNCTIONS
  - Parameters
  - Return
  - Calling them

READING

- Connolly/Begg (4th ed) 8.2.4
- Earp/Bagui Ch. 12, 13
- Shah Part 3 (Ch 10,12)
- Morrison/Morrison Ch.4, 9 – selected bits
- Casteel, J (2003). Oracle 9i Developer: PL/SQL Programming