PL/SQL Control Structures

See http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96624/04_struc.htm

Iteration

Loop …Exit, While..Loop, For..Loop

Iterative Control:

- Why are loops needed?
  - To repeat the same group of actions:
    - Infinitely
    - While a condition is true
    - For every element in a set
    - For a specific number of times
- Common use of loops in PL/SQL
  - Setting up an infrastructure in a 1:many situation
  - Retrieving and manipulating multiple rows.

Using a FOR loop…

- I'm organising a tournament with 32 teams. I need to set up matches:
  - 16 first round, 8 second round, 4 quarter finals, 2 semi-finals and 1 final.
  - I can insert partially filled rows using a FOR loop.
- I'm introducing a new programme in the DIT. I need to set up 4 stages:
  - Use a FOR loop to insert the partially filled stage rows

Outline – FOR loop

- Find out the programme code, programme name and chairperson's name.
- Find out the number of stages in the programme.
- Assume we will allow 40 students per stage.
- Pseudocode:
  - Get input
  - Insert programme row
  - For i = 1 to <number of stages>, add a stage row.

Add a new programme

- Add a programme

FOR Loop (1 of 3) Declarations

```
DECLARE
   V_PRCODE programme.prog_code%TYPE :=
    '&Enter_Programme_Code';
   V_CHAIR programme.course_chairperson%TYPE :=
'&Enter_chairperson_Name';
   V_PGNAME programme.prog_name%TYPE :=
    '&Enter_Programme_Name';
   V_NSTG integer := &Enter_Number_of_Stages;
```

FOR Loop (2 of 3) Executable

```
BEGIN
   INSERT INTO PROGRAMME (
    prog_code, prog_name, course_chairperson)
   VALUES (V_PRCODE, V_PGNAME, V_CHAIR);
  FOR i IN 1..V_NSTG
  LOOP
     INSERT INTO STAGE(
     prog_code, stage_code, remaining_places)
     VALUES (V_PRCODE,i,40);
  END LOOP;
 commit;
```

FOR loop (3 of 3) Exceptions

```
EXCEPTION
WHEN OTHERS THEN
      ROLLBACK WORK;
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
end;
```

Cursors

How to step through the returned set of rows

Handling Multiple Rows

- To set aside a mechanism for holding multiple rows, we can use a CURSOR.
- A CURSOR allows the user to define an area in memory to hold the results of a SELECT statement.

Managing explicit cursors

- In the declarations:
  - Define the cursor
  - Set aside a mechanism for holding a cursor row.
- In the executable section:
  - OPEN the cursor
  - FETCH from the cursor
  - CLOSE the cursor.

Declaring a simple cursor

CURSOR cursor_name IS select_statement;

- **Example**

```
DECLARE
  CURSOR inprofit IS
    SELECT stock_code, unit_price
    from stock
    where (unit_price>unitcostprice);
```

- This cursor will contain a set of rows, each with two attributes, a stock_code and a unit_price.
- The scope of the cursor is the scope of the block in which it is declared.

Where will the result set go?

- The SELECT statement will return a result set.
- The next item to declare is a *row holder* for the result set.
- This can be based on the data types in the tables from which the cursor is pulling rows:
- *rsetname* *<cursorname>*%rowtype;
- e.g. For the cursor called inprofit, we can call the resulting rows IP_REC:

```
IP_REC inprofit%rowtype;
```

Our declarations:

```
DECLARE
-- cursor
CURSOR inprofit IS
     SELECT stock_code, unit_price
     from stock
     where (unit_price > unitcostprice);
-- holder for returned row
IP_REC inprofit%ROWTYPE;
```

Opening and closing the cursor

```
DECLARE
   CURSOR  <cursorname> IS SELECT …
…
BEGIN
   OPEN <cursorname>;
…
   CLOSE <cursorname>;
END;
```

- This open and close the cursor, but don't fill the rows.

Fetching the cursor

- The cursor can fetch rows ***one at a time*** and store them in the cursor row.
  - This is different from the SELECT, in that it cannot do 'one at a time'.
- As the rows are fetched one at a time, the program is required to *iterate* through them.
- Iteration is facilitated by LOOPs.

Fetching

- Fetch retrieves the rows one at a time from the result set.
- The parameters relating to cursors are:

```
%found -- true if something is returned.
%notfound -- true if nothing is returned.
%rowcount -- holds the number of rows returned.
%isopen -- this is true if the cursor is open.
```

Cursor loop

```
LOOP
FETCH inprofit into IP_REC;
EXIT WHEN inprofit%NOTFOUND;
dbms_output.put_line
('Stock code '||IP_REC.stock_code||' at
  '||IP_REC.unit_price||' makes profit.');
END LOOP;
```

- This program fetches rows from the stock table that are profitable and displays them on the screen.

The whole program

```
DECLARE
CURSOR inprofit IS
    SELECT stock_code, unit_price from stock
    where (unit_price > unitcostprice);
IP_REC inprofit%ROWTYPE;
BEGIN
OPEN inprofit;
LOOP
FETCH inprofit into IP_REC;
EXIT WHEN inprofit%NOTFOUND;
dbms_output.put_line
('Stock code '||IP_REC.stock_code||' at
  '||IP_REC.unit_price||' makes profit.');
End Loop;
Close inprofit;
END;
```

Is there another way to do this?

EXIT-WHEN summary

- The EXIT-WHEN statement lets a loop complete conditionally.
- When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement after the loop.

WHILE-LOOP

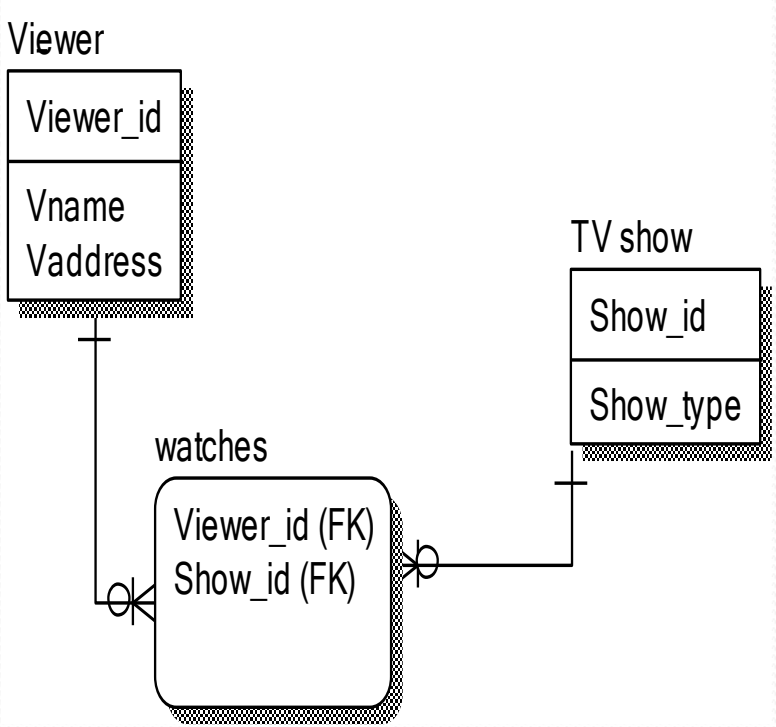
- The WHILE-LOOP statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP, as follows:

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

- *You may investigate this further yourselves.*

Another way of using cursors

- I have a list of TV viewers and a list of TV programmes.
- I want to indicate that a viewer watches all crime shows (i.e. Show_type = 'Crime').
- I can use a cursor.



Outline plan

- I need to take in a viewer's id.
  - *What if the viewer isn't recorded? I can use an exception.*
- I need to return the shows that are crime shows.
  - *I can use a cursor*
- I need to record 'watches' for each crime show for this viewer.
  - *What if the viewer already watches crime shows?*
- I need to report on the number of rows added.
  - *I can use my cursor statistics.*
- What if something unexpected happens?
  - *Exceptions*

AddCrimeViewer(1 of 3)

```
/*Add a viewer that watches only crime shows*/
SET SERVEROUTPUT ON
DECLARE
  V_VIEWER_ID viewer.viewer_id%type :=
                    '&Enter_Viewer_id';
  CURSOR crimes IS
    SELECT show_id FROM tv_show
         WHERE show_type LIKE 'Crime';
  thisshow tv_show.show_id%type;
  V_VNAME      viewer.vname%type;
```

AddCrimeViewer(2 of 3)

```
BEGIN
  SELECT vname INTO V_VNAME FROM viewer WHERE
 viewer_id LIKE V_VIEWER_ID;
  OPEN crimes;
  LOOP
    FETCH crimes INTO thisshow;
    EXIT WHEN crimes%notfound;
    INSERT INTO watches VALUES(V_VIEWER_ID,
 thisshow);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('There were '||
 crimes%rowcount||' crime shows.');
  CLOSE CRIMES;
  commit;
```

AddCrimeViewer(3 of 3)

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('This viewer does not exist.');
  ROLLBACK WORK;
WHEN DUP_VAL_ON_INDEX THEN
  DBMS_OUTPUT.PUT_LINE('A viewer was found that
  already watches crime - the operation has failed.');
  ROLLBACK WORK;
WHEN  OTHERS THEN
  DBMS_OUTPUT.PUT_LINE ('The error '||SQLCODE||' has
  occurred.'||' meaning '||SQLERRM||'.');
  ROLLBACK WORK;
END;
```

Demo

- Record that a viewer watches all crime shows.