(1)

Store Procedure/Function  
PL/SQL

Part 2

OPEN UP ORACLE

Then you can cut and paste and try these code examples

# AGENDA - topics

(2)

1 Conditional Statements

2 Iteration Statements

3 Cursors

4 User-defined Functions

# Conditional Statements -- Repeat/Review

(3)(4 syntax)

The search CASE statement executes the statement of the first true condition.

The remaining conditions are not evaluated after the first true condition.

Syntax

**CASE**

**WHEN** condition\_1 **THEN** statements

**WHEN** condition\_2 **THEN** statements

...

**WHEN** condition\_n **THEN** statements

[ **ELSE**

statements ]

**END** **CASE**;

Previous example:

(6)

SET SERVEROUTPUT ON; 🡸 reminder

**DECLARE**

semester **CHAR**(1);

**BEGIN**

semester := 'S’; -- defined semester with value S

**CASE**

**WHEN** semester = 'F' **THEN** DBMS\_OUTPUT.PUT\_LINE('Fall Term');

**WHEN** semester = 'W' **THEN** DBMS\_OUTPUT.PUT\_LINE('Winter Term');

**WHEN** semester = 'S' **THEN** DBMS\_OUTPUT.PUT\_LINE('Summer Term');

**ELSE** DBMS\_OUTPUT.PUT\_LINE ('Wrong Value');

**END CASE**;

**END**;

BUT … what if it is none of the choices **Do an exception**

DECLARE

semester CHAR(1);

BEGIN

semester := 'J'; -- change choice

CASE

WHEN semester = 'F' THEN DBMS\_OUTPUT.PUT\_LINE('Fall Term');

WHEN semester = 'W' THEN DBMS\_OUTPUT.PUT\_LINE('Winter Term');

WHEN semester = 'S' THEN DBMS\_OUTPUT.PUT\_LINE('Summer Term');

--ELSE DBMS\_OUTPUT.PUT\_LINE ('Wrong Value'); remove this line

END CASE;

EXCEPTION

WHEN CASE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No Semester Found');

END;

2 Iteration Statements

(7)

# LOOP Statements

(8)

Same as any other programming language

Same as SELECT in SQL in what it does -- it loops

LOOP and variations to loop processing

* A LOOP statements runs a series of statements multiple times.
  + **Basic** LOOP
  + **FOR** LOOP
  + **Cursor FOR** LOOP
  + **WHILE** LOOP
* Statement or conditions to exit a loop:
  + EXIT
  + EXIT WHEN
* The statements that exits the current iteration of a loop only and skips to the next iteration.
  + CONTINUE
  + CONTINUE WHEN

# Basic LOOP

(9)

The loop executes the statements until an EXIT statement terminates the loop execution or an exception is raised.

Just like other programming languages

The EXIT statement terminates the loop and transfers the control to the end of the current loop

Look at a sample

**DECLARE**

counter **NUMBER** := 3; -- setting counter to 3

BEGIN

DBMS\_OUTPUT.PUT\_LINE ('---- Count Down -----'); -- put up a title

**LOOP**

DBMS\_OUTPUT.PUT\_LINE ('COUNTER #: ' || counter); -- shows loop value

counter := counter - 1;

**IF** counter < 1 **THEN**

Exiting out of a loop when you know the condition was considered bad form

EXIT;

**END IF**;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE('End of the LOOP!');

**END**;

---- Count Down -----

EXTRA NOTE: If procedure is not "created" and compiled it will run. Other wise you need a BEGIN run it and END

COUNTER #: 3

COUNTER #: 2

COUNTER #: 1

End of the LOOP!

# EXIT WHEN

(10)

The test is at the end, so it always enters the loop once

**DECLARE**

counter **NUMBER** := 5;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE ('---- Count Down -----');

**LOOP**

DBMS\_OUTPUT.PUT\_LINE ('counter: ' || counter);

counter := counter - 1;

**EXIT WHEN** counter < 3;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE('End of the LOOP!');

**END**;

---- Count Down -----

counter: 5

counter: 4

counter: 3

End of the LOOP!

C equivalent – do … while

# NESTED LOOPS

(11)

A LOOP statement can be inside another LOOP statement.

The EXIT statement inside the inner LOOP exits the inner LOOP

- and transfers the control to the outer loop.

My favourite i and j variables

**DECLARE**

i **NUMBER** := 0;

j **NUMBER** := 2;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE('Beginning of the Code!');

**LOOP**

i := i + 1;

DBMS\_OUTPUT.PUT\_LINE ('---- i: ' || i); -- I is now 1 first time

j:= 3;

**LOOP** -- enter the inner loop

DBMS\_OUTPUT.PUT\_LINE ('-- j: ' || j); -- j is 1 the first time

j := j - 1; -- j increments by 1 and is now 2

**EXIT WHEN** j < 0; -- it is not less than one , stay in inner loop

**END LOOP**;

**EXIT WHEN** i > 1;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE('End of the Code!');

**END**;

The output

Beginning of the Code!

---- i: 1

There is nothing new here. Once you learn one or two languages the others are vey similar.

-- j: 3

-- j: 2

-- j: 1

-- j: 0

---- i: 2

-- j: 3

-- j: 2

-- j: 1

-- j: 0

End of the Code!

# CONTINUE

(12)

The CONTINUE statement exits the current iteration of the loop and goes to the next iteration.

In the example: The following code does not output value 2 for the variable counter.

**DECLARE**

counter **NUMBER** := 4;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE ('---- Count Down -----');

**LOOP**

counter := counter - 1;

**IF** counter = 2 **THEN**

**CONTINUE**;

**END IF**;

DBMS\_OUTPUT.PUT\_LINE ('counter: ' || counter);

**EXIT WHEN** counter < 1;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE('End of the LOOP!');

**END**;

---- Count Down -----

Drops out of the rest of the loop but continues back in the loop

counter: 3

counter: 1

counter: 0

End of the LOOP!

# CONTINUE WHEN

Looks to do the same thing.

**DECLARE**

counter **NUMBER** := 4;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE ('---- Count Down -----');

**LOOP**

counter := counter - 1;

**IF** counter = 2 **THEN**

**CONTINUE**;

**END IF**;

**CONTINUE WHEN** counter = 2;

DBMS\_OUTPUT.PUT\_LINE ('counter: ' || counter);

**EXIT WHEN** counter < 1;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE('End of the LOOP!');

**END**;

counter: 3

counter: 1

counter: 0

End of the LOOP!

# FOR LOOP

(14)

Again similar

The FOR LOOP statement executes the statements inside the loop while the value of the loop index is in a given range.

DEFAULT  
starts at lower number and increments by 1 until upper condition met.

IF you include the REVERSE keyword, the value of index starts from the upper bound value and deceases by one until it becomes equal to the lower bound value.

Of course, the upper bound value must be greater than or equal to the lower bound value.

Index is the local variable of the FOR loop.

SYNTAX

**FOR** index **IN** [ REVERSE ] lower\_bound … upper\_bound **LOOP**

statements

**END** **LOOP**;

# EXAMPLE: FOR LOOP

(15)

Can space it 1 .. 4

**BEGIN**

**FOR** i **IN** 1..4 **LOOP**

**IF** i < 2 **THEN**

DBMS\_OUTPUT.PUT\_LINE ( i || ' is less than 2');

**ELSIF** i > 2 **THEN**

DBMS\_OUTPUT.PUT\_LINE ( i || ' is greater than 2');

**ELSE**

DBMS\_OUTPUT.PUT\_LINE ( i || ' is equal to 2');

**END IF**;

**END LOOP**;

**END**;

OUTPUT:

1 is less than 2

2 is equal to 2

3 is greater than 2

4 is greater than 2

# NESTED FOR LOOPS

(16)

Same idea as any language

**BEGIN**

**FOR** x **IN** 1 .. 2 **LOOP**

DBMS\_OUTPUT.PUT\_LINE ('---- x: ' || x );

**FOR** y **IN REVERSE** 1 .. 4 **LOOP**

DBMS\_OUTPUT.PUT\_LINE ('-- y: ' || y );

**END LOOP**;

**END LOOP**;

**END**;

OUTPUT because I and j are harder to see I switched to x and y

---- x: 1

-- y: 4

-- y: 3

-- y: 2

-- y: 1

---- x: 2

-- y: 4

-- y: 3

-- y: 2

-- y: 1

# WHILE LOOP

(18)

The WHILE executes if the condition is TRUE.

It stops when FALSE or an EXIT

Control passes to the statement after the WHILE loop

**DECLARE**

run **BOOLEAN** := true;

round **NUMBER** := 1;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE ('-- First WHILE LOOP --');

**WHILE** run **LOOP**

DBMS\_OUTPUT.PUT\_LINE ('round ' || round);

round := round + 1;

**IF** round = 4 **THEN**

run := false;

**END IF**;

**END LOOP**;

DBMS\_OUTPUT.PUT\_LINE ('-- Second WHILE LOOP --');

**WHILE NOT** run **LOOP**

DBMS\_OUTPUT.PUT\_LINE ('round ' || round);

round := round - 1;

**IF** round = 0 **THEN**

run := true;

**END** **IF**;

**END LOOP**;

**END**;

Can also use an ordinary loop control instead of boolean

DECLARE

round NUMBER := 1;

BEGIN

DBMS\_OUTPUT.PUT\_LINE ('-- First WHILE LOOP --');

WHILE round <5 LOOP

DBMS\_OUTPUT.PUT\_LINE ('round ' || round);

round := round + 1;

END LOOP;

END;

-- First WHILE LOOP --

round 1

round 2

round 3

-- Second WHILE LOOP --

round 4

round 3

round 2

round 1

CURSORS  
(19 - 20)

Cursors are used to process multiple rows in PL/SQL blocks.

In this course, we learn fundamentals about cursors.

We use cursors to return multiple rows from a PL/SQL procedure to a caller procedure or program.

Lots of words…. Let us see what it means

# PL/SQL CURSORS

(21)

A cursor is a pointer to a context area that includes the result of a processed SQL statement.

Translation: Simply, a cursor contains the rows of a select statement.

In PL/SQL, cursors are used to access and process the rows returned by a SELECT statement.

There are two types of cursors:

* + Implicit cursors
  + Explicit cursors

IMPLICIT CURSOR (22)

One that is not defined … implied

We do not have this table. Will need to be improved

Go to next page

## Implicit Cursor Attributes

Following are implicit cursor attributes,

|  |  |  |
| --- | --- | --- |
| Cursor Attribute | Cursor Variable | Description |
| %ISOPEN | SQL%ISOPEN | Oracle engine automatically open the cursor If cursor open **return TRUE** otherwise **return FALSE.** |
| %FOUND | SQL%FOUND | If SELECT statement return one or more rows or DML statement (INSERT, UPDATE, DELETE) affect one or more rows If affect **return TRUE** otherwise **return FALSE.** If not execute SELECT or DML statement **return NULL.** |
| %NOTFOUND | SQL%NOTFOUND | If SELECT INTO statement return no rows and fire no\_data\_found PL/SQL exception before you can check SQL%NOTFOUND. If not affect the row **return TRUE** otherwise **return FALSE.** |
| %ROWCOUNT | SQL%ROWCOUNT | Return the number of rows affected by a SELECT statement or DML statement (insert, update, delete). If not execute SELECT or DML statement **return NULL.** |

Sy

Jump to page 20

Using EMP table if loaded in week 6

EMPLOYEE\_ID EMPLOYEE\_N JOB MANAGER\_ID HIREDATE SALARY COMMISSION DEPARTMENT\_ID

----------- ---------- --------- ---------- -------- ---------- ---------- -------------

7369 SMITH CLERK 7902 80-12-17 800 20

7499 ALLEN SALESMAN 7698 81-02-20 1600 300 30

7521 WARD SALESMAN 7698 81-02-22 1250 500 30

7566 JONES MANAGER 7839 81-04-02 2975 20

7654 MARTIN SALESMAN 7698 81-09-28 1250 1400 30

7698 BLAKE MANAGER 7839 81-05-01 2850 30

7782 CLARK MANAGER 7839 81-06-09 2450 10

7788 SCOTT ANALYST 7566 87-04-19 3000 20

7839 KING PRESIDENT 81-11-17 5000 10

7844 TURNER SALESMAN 7698 81-09-08 1500 0 30

7876 ADAMS CLERK 7788 87-05-23 1100 20

7900 JAMES CLERK 7698 81-12-03 950 30

7902 FORD ANALYST 7566 81-12-03 3000 20

7934 MILLER CLERK 7782 82-01-23 1300 10

14 rows selected.

set serveroutput on

BEGIN

UPDATE emp

SET job = 'Web Dev'

WHERE employee\_name='MILLER';

IF SQL%FOUND THEN

dbms\_output.put\_line('Updated - If Found employee');

END IF;

IF SQL%NOTFOUND THEN

dbms\_output.put\_line('NOT Updated - If employee NOT Found');

END IF;

IF SQL%ROWCOUNT>0 THEN

dbms\_output.put\_line(SQL%ROWCOUNT||' Rows Updated');

ELSE

dbms\_output.put\_line('NO Rows were found Updated Found');

END IF;

END;

OUTPUT  
  
Updated - If Found employee

1 Rows Updated

# EXPLICIT CURSOR

(23)

The explicit cursors are defined in the declaration section of a PL/SQL block

Defined by user … programmers.

It is used to process the multi-row results from a SELECT statement.

**Define cursor:**

**CURSOR** cursor\_name **IS** select\_statement;

Go to specifics PP 24

# DECLARE A CURSOR step 1

(24)

Cursors can be defined in the DECLARE section

Format:

**CURSOR** cursor\_name **IS** select\_statement;

**DECLARE**

**CURSOR** cursor\_1 **IS**

Test run just the SQL

This is 20203 script

**SELECT last\_name, job\_id**

**FROM employees**

**WHERE job\_id LIKE 'A%'**

**ORDER BY last\_name;**

LAST\_NAME JOB\_ID

------------------------- ----------

De Haan AD\_VP

Flertjan AC\_REP

Gietz AC\_ACCOUNT

Higgins AC\_MGR

King AD\_PRES

Kochhar AD\_VP

Whalen AD\_ASST

7 rows selected.

# OPEN A CURSOR step 2

(25)

Done in the executable portion. After the BEGIN.

**DECLARE**

e\_last\_name employees.last\_name%type;

e\_job\_tile employees.job\_id%type;

**CURSOR** emp\_cursor **IS**

**SELECT** last\_name, job\_id

**FROM** employees

**WHERE** job\_title **LIKE** 'A%'

**ORDER** **BY** last\_name;

**BEGIN**

**OPEN** emp\_cursor;

# CLOSE A CURSOR step 3

(27 and 28 example)

See example

**DECLARE**

e\_last\_name employees.last\_name%type;

e\_job\_tile employees.job\_id%type;

**CURSOR** emp\_cursor **IS**

**SELECT** last\_name, job\_id

**FROM** employees

**WHERE** job\_id LIKE 'A%'

**ORDER** **BY** last\_name;

**BEGIN**

**OPEN** emp\_cursor;

**LOOP**

**FETCH** emp\_cursor into e\_last\_name, e\_job\_tile;

**EXIT** **WHEN** emp\_cursor%notfound;

dbms\_output.put\_line(e\_last\_name || ' ' || e\_job\_tile);

**END LOOP**;

**CLOSE** emp\_cursor;

**END**;

OUTPUT:

De Haan AD\_VP

Flertjan AC\_REP

Gietz AC\_ACCOUNT

Higgins AC\_MGR

King AD\_PRES

Kochhar AD\_VP

Whalen AD\_ASST

# EXPLICIT CURSORS with parameters

The %ROWTYPE attribute lets you declare a record that represents a row in a table or view

(29)

**DECLARE**

Defined with 2 parameters

p\_product products%rowtype;

**CURSOR** product\_cursor (price\_1 **NUMBER**, price\_2 **NUMBER**)

**IS**

**SELECT** \*

**FROM** products

**WHERE** prod\_sell **BETWEEN** price\_1 **AND** price\_2;

**BEGIN**

**OPEN** product\_cursor (100, 500); -- parameters

**LOOP**

**FETCH** product\_cursor **INTO** p\_product;

**EXIT WHEN** product\_cursor%notfound;

dbms\_output.put\_line(p\_product.prod\_name || ': ‘ ||p\_product.prod\_sell);

**END LOOP**;

**CLOSE** product\_cursor;

**END**;

OUTPUT:

Star Lite: 200

MoonBeam: 120

MoonGlow: 129

Pack n' Hike: 131

Dover-2: 111

Pro-Lite Water Filter: 165

Pocket Water Filter: 270

# EXPLICIT CURSORS with parameters

(30)

FOR LOOPS will open cursor and close the cursor

when no more rows found

**DECLARE**

e\_last\_name employees.last\_name%type;

e\_job\_tile employees.job\_id%type;

**CURSOR** emp\_cursor **IS**

**SELECT** last\_name, job\_id

**FROM** employees

**WHERE** job\_id **LIKE** 'A%'

**ORDER** **BY** last\_name;

**BEGIN**

**FOR** item **IN** emp\_cursor -- begins a FOR loop

**LOOP**

DBMS\_OUTPUT.PUT\_LINE

('NAME = ' || item.last\_name || ', JOB = ' || item.job\_id);

**END LOOP**;

**IF** emp\_cursor%ISOPEN **THEN**

CLOSE emp\_cursor;

**END IF**;

**END**;

NAME = De Haan, JOB = AD\_VP

NAME = Flertjan, JOB = AC\_REP

NAME = Gietz, JOB = AC\_ACCOUNT

NAME = Higgins, JOB = AC\_MGR

NAME = King, JOB = AD\_PRES

NAME = Kochhar, JOB = AD\_VP

NAME = Whalen, JOB = AD\_ASST

# EXPLICIT CURSOR ATTRIBUTES

(31)

Look over these later

|  |  |
| --- | --- |
| **Attributes** | **Value** |
| **%ISOPEN** | **TRUE**: if the cursor is open  **FALSE**: if the cursor is not open |
| **%FOUND** | INVALID\_CURSOR: if the cursor is not open  NULL: before we fetch the first row  FALSE: if the fetch row is successfully  TRUE: if no row is fetched in the fetch statement |
| **%NOTFOUND** | INVALID\_CURSOR: if the cursor is not open.  NULL: before we fetch the first row |
| **%ROWCOUNT** | INVALID\_CURSOR: if the cursor is not open  Otherwise: It returns the number of rows returned from the cursor |

**Adding a bit more**

Put a counter in to see how many rows generated.

DECLARE

cnt NUMBER :=0; -- start a counter

e\_last\_name employees.last\_name%type;

e\_job\_tile employees.job\_id%type;

CURSOR emp\_cursor IS

SELECT last\_name, job\_id

FROM employees

WHERE job\_id LIKE 'A%'

ORDER BY last\_name;

**BEGIN**

FOR item IN emp\_cursor -- begins FOR loop

LOOP

DBMS\_OUTPUT.PUT\_LINE

('NAME = ' || item.last\_name || ', JOB = ' || item.job\_id);

cnt := cnt + 1;

END LOOP;

**IF** cnt >0 THEN

dbms\_output.put\_line(cnt || ' Rows Updated');

**ELSE**

dbms\_output.put\_line('NO Rows were found Updated Found');

**END IF;**

**IF** emp\_cursor%ISOPEN THEN

CLOSE emp\_cursor;

**END IF;**

**END;**

NAME = De Haan, JOB = AD\_VP

NAME = Flertjan, JOB = AC\_REP

NAME = Gietz, JOB = AC\_ACCOUNT

NAME = Higgins, JOB = AC\_MGR

NAME = King, JOB = AD\_PRES

NAME = Kochhar, JOB = AD\_VP

NAME = Whalen, JOB = AD\_ASST

7 Rows Updated

USER-DEFINED FUNCTIONS

(32)

Near the end

# Create a PL/SQL Function

(33)

Generic

**CREATE** **[OR REPLACE] FUNCTION** function\_name (parameter\_list)

**RETURN** return\_type

**IS/AS**

[declarative section]

**BEGIN**

[executable section]

[**EXCEPTION**]

[exception-handling section]

**RETURN** return\_value

**END**;

# PL/SQL Function Example

(34)

**CREATE** **OR REPLACE FUNCTION** find\_max\_price

**RETURN** NUMBER

**IS**

max\_price NUMBER := 0;

**BEGIN**

-- get the maximum prod\_sell price

**SELECT** MAX(prod\_sell)

**INTO** max\_price

**FROM** products;

-- return the max price

**RETURN** max\_price;

**END**;

Run the above just compiles it

It returns maximum list price, **but we never showed what it was.**

# Using Functions in Assignment Statements

(35)

A function returns a value

Assign the value to a variable and use it

**DECLARE**

highest\_price products.prod\_sell%type := 0.0;

**BEGIN**

highest\_price := find\_max\_price(); -- 🡸 call the function

dbms\_output.put\_line('The maximum price is ' || highest\_price); -- 🡸 output the results

**END**;

OUTPUT:

The maximum price is 8867.99

# Using Functions in Conditional Statements

(36)

DECLARE

new\_price products.prod\_sell%type := 9;

BEGIN

IF (new\_price < find\_max\_price()) THEN -- used the function for comparison

dbms\_output.put\_line('The new price is lower than the maximum price.');

ELSE

dbms\_output.put\_line('The new price is higher than the maximum price.');

END IF;

END;

# Use PL/SQL Functions in SQL Statements

(37)

PROBLEM: Company wished to double the price of each product.

Return a list of products where the new doubled price is greater than the current maximum price

EXAMPLE SQL

**SELECT** prod\_no,

prod\_name,

prod\_sell,

(prod\_sell \* 2) as "New Price"

**FROM** products

**WHERE** (prod\_sell \* 2) > find\_max\_price();

Once you built it, you can use it in just SQL only

PROD\_NO PROD\_NAME PROD\_SELL New Price

---------- ------------------------------ ---------- ----------

40101 Star Gazer-2 553 1106

40102 Star Gazer-3 590 1180

40103 StarDome 650 1300

# DROP FUNCTION

DROP FUNCTION function\_name;

THE END

Extra sample:

Showing control over output to get another layout

DECLARE

cursor XX is select \*

from emp

where employee\_id <=7600;

tmp emp%rowtype;

BEGIN

-- OPEN X; -- opened and close by FOR loop

FOR tmp IN XX

LOOP

dbms\_output.put\_line('No: '||tmp.employee\_id);

dbms\_output.put\_line('Name: '||tmp.employee\_name);

dbms\_output.put\_line('Job: '||tmp.job);

dbms\_output.put\_line('Salary:'||tmp.salary);

dbms\_output.put\_line(' --------------------');

END Loop;

-- CLOSE X;

END;

OUTPUT:

No: 7369

Name: SMITH

Job: CLERK

Salary:800

--------------------

No: 7499

Name: ALLEN

Job: SALESMAN

Salary:1600

--------------------

No: 7521

Name: WARD

Job: SALESMAN

Salary:1250

--------------------

No: 7566

Name: JONES

Job: MANAGER

Salary:2975

--------------------