Bases de données

Prof. Giovanna Di Marzo Serugendo
Assistant: Mohammad Parhizkar
mohammad.parhizkar@unige.ch

Semestre: printemps 2019

session 3- 11/03/2019

TP average ....?

We will choose the average of the greatest five TPs.

Example 1:

TP1	TP2	TP3	TP4	TP5	TP6	TP7
0	6	5.5	4	6	6	4.5

$$\frac{6+5.5+6+6+4.5}{5}=5.6$$

Example 2:

TP1	TP2	TP3	TP4	TP5	TP6	TP7
0	0	5	0	6	3	0

$$\frac{5+0+6+3+0}{5}=2.8$$

Modular design in PL/SQL

PL/SQL DBMS Output

The **DBMS_OUTPUT** is a built-in package to display the output of your code.

Example 1:

```
SET SERVEROUTPUT ON;

BEGIN
    dbms_output.put_line('Helloooooooo');
END;
```

a simple PL/SQL anonymous block

Script output 1:

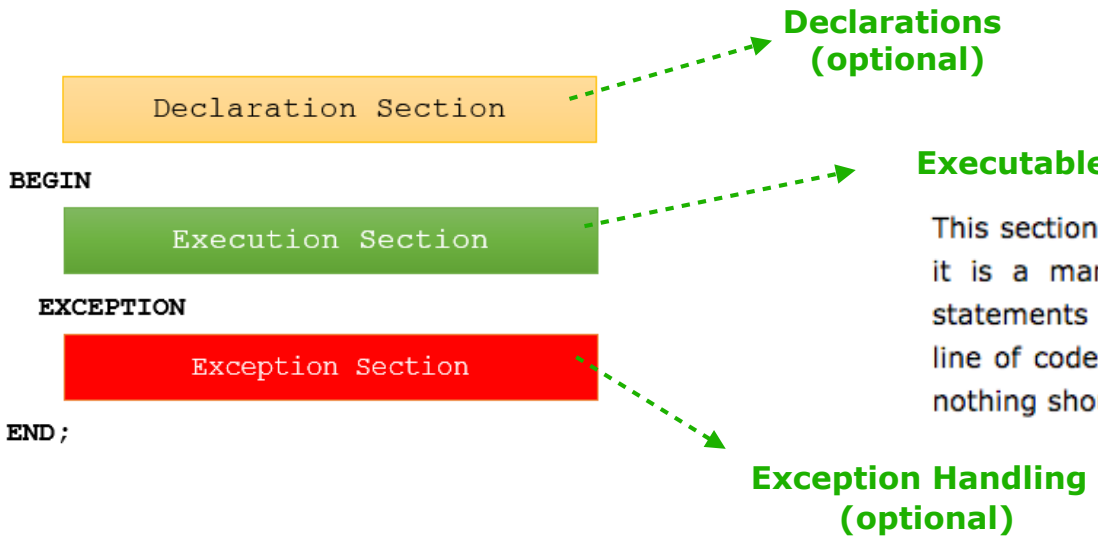
```
Helloooooooo

PL/SQL procedure successfully completed.
```

Blocks

- A block without a name is an anonymous block.
- An anonymous block is not saved in the Oracle Database server.
- Thus, it is just for one-time use.

Blocks

Each block consists of 3 sub-parts:



Executable Commands

This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL command** to indicate that nothing should be executed.

Example:

```
DECLARE
   message  varchar2(20):= 'Today is monday!';
BEGIN
   dbms_output.put_line(message);
END;
/
```

Script Output:

```
Today is monday!

PL/SQL procedure successfully completed.
```

TWO Anonymous Blocks

Worksheet | Query Builder

```
SET SERVEROUTPUT ON;

--example shows 2 anonymous block
BEGIN
DECLARE
   current_date DATE := SYSDATE;
BEGIN
     dbms_output.put_line( current_date );
END;

DECLARE
BEGIN
/* Manipulating the rows ( also called record or tuples) of table directors (from tp1) */

--INSERT a new row
INSERT INTO directors (director_id, director_first_name, director_last_name, director_BD, country) VALUES(11, 'Guillermo',
'Del Toro', TO_DATE('09/11/1954','DD/MM/YYYY'), 'Mexico');
--Delete the same row
DELETE FROM directors WHERE director_first_name= 'Guillermo';
END;
END;
```

ERROR:

Oracle can NOT take more than one PL/SQL anonymous block at a time.

SOLUTIONS:

1. Wrap the two anonymous blocks within another anonymous block (having two sub-blocks).
2. You need a slash `/` on the line after each `end;`.

PL/SQL Math Functions

Oracle PL / SQL / Numeric Math Functions /

Numeric Math Functions /

ABS 8	ACOS 3
ASIN 3	ATAN 2
ATAN2 2	BIN_TO_NUM 4
BITAND 5	CEIL 6
COS 4	COSH 3
EXP 4	FLOOR 4
greatest 6	hextoraw 1
least 4	LN 2
LOG 4	MOD 8
POWER 7	ROUND 19
SIGN 8	SIN 3
SINH 2	SQRT 5
TAN 3	TANH 3
TRUNC 13	

Example:

```
ROUND(345.678) CEIL(345.678) FLOOR(345.678)
-------------- ------------- --------------
           346           346            345
```

http://www.java2s.com/Code/Oracle/Numeric-Math-Functions/CatalogNumeric-Math-Functions.htm

PL/SQL Placeholders

- Placeholders are **temporary** storage area.
- PL/SQL Placeholders can be any of Variables, Constants and Records.
- Number (n,m) , Char (n) , Varchar2 (n) , Date , Long , Long raw, Raw, Blob, Clob, Nclob, Bfile, …

Syntax:

```
variable_name datatype [NOT NULL := value ];
                          or  [:= value ];
```

Example 1:

```
DECLARE

salary  number (6);
```

* “salary” is a variable of datatype number and of length 6.

Example 2:

```
DECLARE

salary number(4);

dept varchar2(10) NOT NULL := “HR Dept”;
```

When a variable is specified as NOT NULL, you must initialize the variable when it is declared.

PL/SQL Variables

from TP1
Example 3:

```
DECLARE
 var_name Varchar2(20);
 var_director_id number(2) := 9;
BEGIN
 SELECT director_first_name
 INTO var_name
 FROM directors
 WHERE director_id = var_director_id;

 dbms_output.put_line('The director '
    || var_director_id || '  name  is ' || var_name);
END;
/
```

variables inside a block

Initializing Variables in PL/SQL

For each item in the **SELECT list**, there must be a corresponding, type-compatible variable in the **INTO list**.

Script output:

```
The director 9  name  is Steven


PL/SQL procedure successfully completed.
```

DIRECTOR_ID	DIRECTOR_FIRST_NAME	DIRECTOR_LAST_NAME	DIRECTOR_BD	COUNTRY
1	Ivan	Reitman	01.03.46	Slovakia
2	Joseph	Mankiewicz	01.03.09	USA
3	John	Lasseter	01.03.57	USA
4	Pete	Doctor	(null)	USA
5	James	Cameron	01.03.54	Canada
6	Paul	Haggis	01.03.53	Canada
7	Brad	Bird	(null)	USA
8	Henry	Hobson	01.03.85	USA
9	Steven	Soderbergh	01.03.63	USA
10	Jason	Reitman	(null)	Canada

PL/SQL Variables

Example 4:

```
SET SERVEROUTPUT ON;

-- TABLE: financial
DECLARE
  total_sales financial.unitssold%type;
 BEGIN
 SELECT SUM(financial.unitssold) INTO total_sales
 FROM financial
 WHERE financial.SEGMENTS= 'Government';
 dbms_output.put_line ('Total amount of sold units for Goverment is:'|| total_sales);
END;
```

total_sales

variables inside a block: same type as column

financial.unitsold

Script output:

```
Total amount of sold units for Goverment is:470673.5


PL/SQL procedure successfully completed.
```

%TYPE is used to declare variables with relation to the data type of a column in an existing table.

file: IFTHEN-example.sql

PL/SQL Variables

from TP1

Example 5:

```
SET Serveroutput On;
DECLARE
   c_id directors.director_id%type := 5;
   c_name  directors.director_first_name%type;
   c_last_name  directors.director_last_name%type;
   c_sal  directors.director_salary%type;
BEGIN
   SELECT director_first_name,director_last_name, director_salary INTO c_name, c_last_name, c_sal
   FROM directors
   WHERE directors.director_id = c_id;
   dbms_output.put_line
   ('Direcotr ' ||c_name  || ' ' || c_last_name  || ' earns ' || c_sal);
END;
/
```

Initializing Variables in PL/SQL

1 2 3 1 2 3

Script output:

```
Direcotr James Cameron earns 101000


PL/SQL procedure successfully completed.
```

%TYPE is used to declare variables with relation to the data type of a column in an existing table.

PL/SQL IF THEN statement

Syntax:

```
IF condition_1 THEN
  statements_1
ELSIF condition_2 THEN
  statements_2
[ ELSIF condition_3 THEN
    statements_3
]
...
[ ELSE
    else_statements
]
END IF;
```

PL/SQL IF THEN statement

Example:

```
SET SERVEROUTPUT ON;

-- TABLE: financial

DECLARE
    total_sales     financial.unitssold%TYPE := 0;
    result_status   VARCHAR2(20);
BEGIN
    SELECT
        SUM(financial.unitssold)
    INTO
        total_sales
    FROM
        financial
    WHERE
        financial.segments = 'Government';

    dbms_output.put_line('Total amount of sold units for Goverment is: '|| total_sales);
 --------
    IF
        total_sales > 400000
    THEN
        result_status := 'High';
        dbms_output.put_line('The result is  '|| result_status);
    ELSIF total_sales = 400000 THEN
        result_status := 'Ok';
        dbms_output.put_line('The result is  ' || result_status);
    ELSE
        result_status := 'low';
        dbms_output.put_line('The result is  ' || result_status);
    END IF;

END;
```

Script output:

```
Total amount of sold units for Goverment is: 470673.5
The result is  High


PL/SQL procedure successfully completed.
```

file: IFTHEN-example.sql

PL/SQL FOR loop

Syntax:

```
FOR index IN lower_bound .. upper_bound
    LOOP
    statements;
END LOOP;
```

Example 1:

block

```
BEGIN
  FOR n_counter IN 1..5
  LOOP
    DBMS_OUTPUT.PUT_LINE( n_counter );
  END LOOP;
END;
```

Result 1:

```
1
2
3
4
5

PL/SQL procedure successfully completed.
```

PL/SQL FOR loop

Example 2:

block

```
DECLARE
  i_counter PLS_INTEGER := 25;
BEGIN
  FOR i_counter IN 1.. 6 loop
    DBMS_OUTPUT.PUT_LINE (i_counter);
  end loop;
  -- after the loop
  DBMS_OUTPUT.PUT_LINE (i_counter);
END;
```

- **PLS_INTEGER**, data type stores signed integers in the range -2,147,483,648 through 2,147,483,647, represented in 32 bits.
- Faster than **INTEGER** and **NUMBER**

Result 2:

```
1
2
3
4
5
6
25
```

i_counter inside the loop

outside the loop

```
PL/SQL procedure successfully completed.
```

PL/SQL FOR loop

Example 3:

```
BEGIN
  FOR z_index IN 1..6 loop
    DBMS_OUTPUT.PUT_LINE (z_index);
  END LOOP;
  -- referencing index after the loop
  DBMS_OUTPUT.PUT_LINE (z_index);
END;
```

ERROR:

Because it references the loop index, which is undefined, outside the FOR LOOP statement.

PL/SQL FOR loop

FOR LOOP with REVERSED keyword

Syntax:

```
FOR index IN REVERSED lower_bound .. upper_bound
    LOOP
    statements;
END LOOP;
```

Example 4:

```
BEGIN
  FOR j_counter IN REVERSE 1..7
  LOOP
    DBMS_OUTPUT.PUT_LINE( j_counter );
  END LOOP;
END;
```

Script output:

```
7
6
5
4
3
2
1


PL/SQL procedure successfully completed.
```

Loop through table and update values in PL/SQL

Example:

part 1

```
ALTER TABLE directors ADD director_salary number(8);
 DECLARE
    updateSalary CONSTANT number(8) := 20000;
BEGIN
    FOR REC IN (SELECT director_id, director_first_name, director_last_name,
    director_bd, country, director_salary, ROWNUM From directors)
    LOOP

            UPDATE  directors SET director_salary = (director_id * updateSalary);

    END LOOP;
END;
 /
```

Adding a new column to directors table

Filling up the table by numbers

20000
40000
60000
…

part 2

```
DECLARE
 salary_increase CONSTANT number(3) :=100;
BEGIN
FOR REC IN (SELECT director_id, director_first_name, director_last_name,
    director_bd, country, director_salary, ROWNUM From directors)
    LOOP

            UPDATE  directors SET director_salary = director_salary + salary_increase;

    END LOOP;
END;
```

Adding a CONSTANT value to all of the rows

Result:

DIRECTOR_ID	DIRECTOR_FIRST_NAME	DIRECTOR_LAST_NAME	DIRECTOR_BD	COUNTRY	DIRECTOR_SALARY
1	Ivan	Reitman	01.03.46	Slovakia	21000
2	Joseph	Mankiewicz	01.03.09	USA	41000
3	John	Lasseter	01.03.57	USA	61000
4	Pete	Doctor	(null)	USA	81000
5	James	Cameron	01.03.54	Canada	101000
6	Paul	Haggis	01.03.53	Canada	121000
7	Brad	Bird	(null)	USA	141000
8	Henry	Hobson	01.03.85	USA	161000
9	Steven	Soderbergh	01.03.63	USA	181000
10	Jason	Reitman	(null)	Canada	201000

PL/SQL WHILE loop

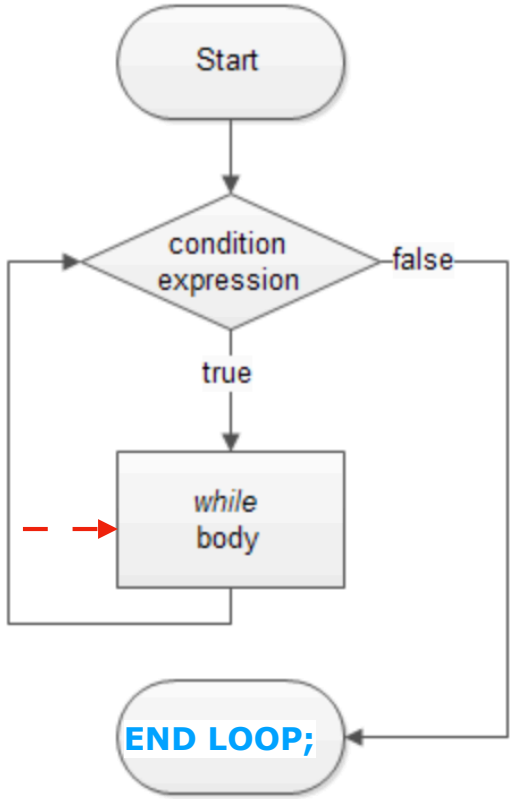
PL/SQL WHILE loop statement to execute a sequence of statements as long as a specified condition is TRUE

Syntax:

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

Boolean: TRUE or FALSE

WHILE loop …



PL/SQL WHILE loop

Example:

Factorial function:

```
n! = n*(n-1)!
   = n*(n-1)*(n-2)!
      ...
   = n*(n-1)*(n-2)*(n-3)... 1
```

3!

2!

1!

4! = 4 × 3 × 2 × 1

product of all positive integers equal and less than 4

= 24

```
SET SERVEROUTPUT ON;
DECLARE
  i_counter   NUMBER := 6;
  i_factorial NUMBER := 1;
  i_temp      NUMBER;
BEGIN
  i_temp := i_counter;
  WHILE i_counter > 0
  LOOP
    i_factorial := i_factorial * i_counter;
    i_counter   := i_counter - 1;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('factorial of ' || i_temp ||
                       ' is ' || i_factorial);

END;
/
```

Script output:

```
factorial of 6 is 720


PL/SQL procedure successfully completed.
```

PL/SQL CASE Statement

Similar functionality of an IF-THEN-ELSE statement

Example:

```
SET SERVEROUTPUT ON;
DECLARE
  n_pct    employees.commission_pct%TYPE;
  v_eval   varchar2(10);
  n_emp_id employees.employee_id%TYPE := 145;
BEGIN
  -- get commission percentage
  SELECT commission_pct
  INTO n_pct
  FROM employees
  WHERE employee_id = n_emp_id;

  -- evalutate commission percentage
  CASE n_pct
    WHEN 0 THEN
      v_eval := 'N/A';
    WHEN 0.1 THEN
      v_eval := 'Low';
    WHEN 0.4 THEN
      v_eval := 'High';
    ELSE
      v_eval := 'Fair';
  END CASE;
  -- print commission evaluation
  DBMS_OUTPUT.PUT_LINE('Employee ' || n_emp_id ||
                       ' commission ' || TO_CHAR(n_pct) ||
                       ' which is '   || v_eval);
END;
/
```

Script output:

```
Employee 145 commission .4 which is High

PL/SQL procedure successfully completed.
```

Employee table:

144	Peter	Vargas	PVARGAS	650.121.2004	09.07.98	ST_CLERK	2500	(null)	124	50
145	John	Russell	JRUSSEL	011.44.1344.429268	01.10.96	SA_MAN	14000	0.4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05.01.97	SA_MAN	13500	0.3	100	80
147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10.03.97	SA_MAN	12000	0.3	100	80
148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15.10.99	SA_MAN	11000	0.3	100	80

PL/SQL ANY, ALL

ANY and ALL are Multiple row operators.

Example for ALL:

```
SELECT director_last_name
FROM directors
WHERE director_salary > ALL (21000, 81000);
```

AND

Script output:

```
DIRECTOR_LAST_NAME
--------------------
Cameron
Haggis
Bird
Hobson
Soderbergh
Reitman

6 rows selected.
```

Example for ANY:

```
SELECT director_last_name
FROM directors
WHERE director_salary > ANY (21000, 81000);
```

OR

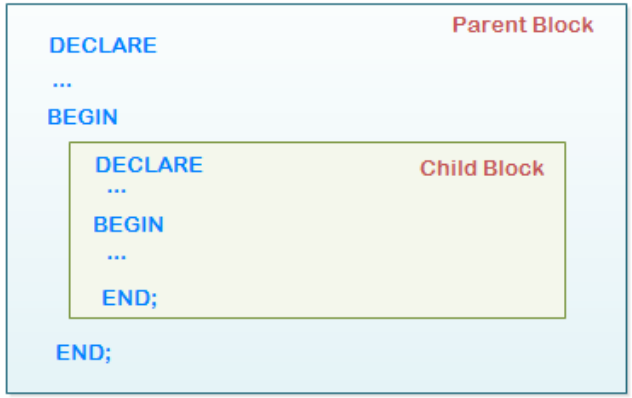
Script output:

```
DIRECTOR_LAST_NAME
--------------------
Mankiewicz
Lasseter
Doctor
Cameron
Haggis
Bird
Hobson
Soderbergh
Reitman

9 rows selected.
```

DIRECTOR_ID	DIRECTOR_FIRST_NAME	DIRECTOR_LAST_NAME	DIRECTOR_BD	COUNTRY	DIRECTOR_SALARY
1	Ivan	Reitman	01.03.46	Slovakia	21000
2	Joseph	Mankiewicz	01.03.09	USA	41000
3	John	Lasseter	01.03.57	USA	61000
4	Pete	Doctor	(null)	USA	81000
5	James	Cameron	01.03.54	Canada	101000
6	Paul	Haggis	01.03.53	Canada	121000
7	Brad	Bird	(null)	USA	141000
8	Henry	Hobson	01.03.85	USA	161000
9	Steven	Soderbergh	01.03.63	USA	181000
10	Jason	Reitman	(null)	Canada	201000

Nested Blocks

- PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**.
- To nest a block means to embed one or more PL/SQL blocks inside another PL/SQL block.



Example 1:

```
 declare
v_title films.title%type;
   begin
     v_title := 'Easy Oracle SQL';  -- in scope

     declare -- a nested block
       v_title films.title%type;
     begin
       v_title := 'Easy Oracle PL/SQL';  -- in scope
      dbms_output.put_line (v_title);  -- in scope    1
    end;

    dbms_output.put_line (v_title); -- in scope     2
  end;
 /
```

We have two variables with the same name v_title

Script output:

```
Easy Oracle PL/SQL     1
Easy Oracle SQL        2

PL/SQL procedure successfully completed.
```

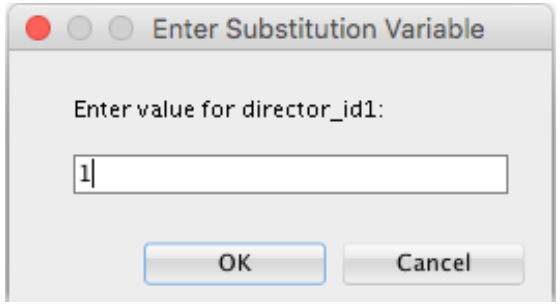
Nested Blocks

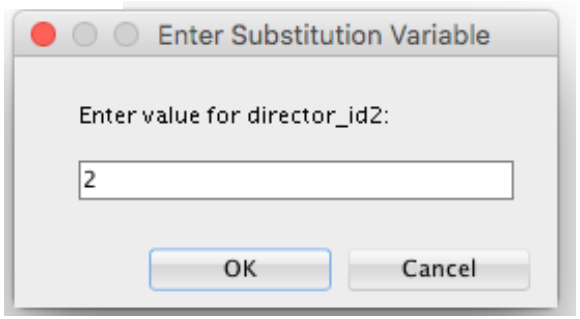
We have two variables with the same name **n_director_id** in the declaration section of both parent and child blocks. The question here is which variable does the SELECT statement accepts?

From TP1

Example 2:

```
SET SERVEROUTPUT ON;
DECLARE
  n_director_id directors.director_ID%TYPE := &director_id1; -- We enter 1 here
BEGIN
  DECLARE
    n_director_id directors.director_id%TYPE := &director_id2; --We enter 2 here
    v_name   directors.director_first_name%TYPE;
  BEGIN
    SELECT director_first_name
    INTO v_name
    FROM directors
    WHERE director_id = n_director_id;

    DBMS_OUTPUT.PUT_LINE('First name of director ' ||  n_director_id ||
                         ' is ' || v_name);

    EXCEPTION
      WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('director ' || n_director_id || ' not found');
  END;
END;
/
```





Nested Blocks

Script output of example 2:

```
new:DECLARE
  n_director_id directors.director_ID%TYPE := 1; -- We enter 1 here
BEGIN
  DECLARE
    n_director_id directors.director_id%TYPE := 2; --We enter 2 here
    v_name   directors.director_first_name%TYPE;
  BEGIN
    SELECT director_first_name
    INTO v_name
    FROM directors
    WHERE director_id = n_director_id;

    DBMS_OUTPUT.PUT_LINE('First name of director ' ||  n_director_id ||
                         ' is ' || v_name);

    EXCEPTION
      WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('director ' || n_director_id || ' not found');
  END;
END;
First name of director 2 is Joseph


PL/SQL procedure successfully completed.
```

(From TP1)

DIRECTOR_ID	DIRECTOR_FIRST_NAME	DIRECTOR_LAST_NAME	DIRECTOR_BD	COUNTRY
1	Ivan	Reitman	01.03.46	Slovakia
2	Joseph	Mankiewicz	01.03.09	USA
3	John	Lasseter	01.03.57	USA
4	Pete	Doctor	(null)	USA
5	James	Cameron	01.03.54	Canada
6	Paul	Haggis	01.03.53	Canada
7	Brad	Bird	(null)	USA
8	Henry	Hobson	01.03.85	USA
9	Steven	Soderbergh	01.03.63	USA
10	Jason	Reitman	(null)	Canada

Nested Blocks: LABELS

a simple example of using block label

```
<<label>>
DECLARE
  v_name varchar2(25) := 'Maria';
BEGIN
  DBMS_OUTPUT.PUT_LINE(label.v_name);
END;
/
```

Nested Blocks: LABELS

We have two variables with the same name **n_director_id** in the declaration section of both parent and child blocks. The question here is which variable does the SELECT statement accepts?

From TP1

Example 3:

```
SET SERVEROUTPUT ON;
<<parent>>
DECLARE
  n_director_id directors.director_ID%TYPE := &director_id1; -- We enter 1 here
BEGIN
  <<child>>
  DECLARE
    n_director_id directors.director_id%TYPE := &director_id2; --We enter 2 here
    v_name   directors.director_first_name%TYPE;
  BEGIN
    SELECT director_first_name
    INTO v_name
    FROM directors
    WHERE director_id = parent.n_director_id;

    DBMS_OUTPUT.PUT_LINE('First name of director ' || parent.n_director_id ||
                         ' is ' || child.v_name);

    EXCEPTION
      WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('director ' || parent.n_director_id || ' not found');
  END;
END;
/
```

Nested Blocks

Script output of example 2:

```
new:<<parent>>
DECLARE
  n_director_id directors.director_ID%TYPE := 1; -- We enter 1 here
BEGIN
  <<child>>
  DECLARE
    n_director_id directors.director_id%TYPE := 2; --We enter 2 here
    v_name   directors.director_first_name%TYPE;
  BEGIN
    SELECT director_first_name
    INTO v_name
    FROM directors
    WHERE director_id = parent.n_director_id;

    DBMS_OUTPUT.PUT_LINE('First name of director ' || parent.n_director_id ||
                         ' is ' || child.v_name);

    EXCEPTION
      WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('director ' || parent.n_director_id || ' not found');
  END;
END;
First name of director 1 is Ivan


PL/SQL procedure successfully completed.
```

PL/SQL Subprogram

A **subprogram** is a program unit/module that performs a particular task.

Subprograms: Blocks, Trigger , Procedures, Functions

Procedures: These subprograms do not return a value directly; mainly used to perform an action.

Functions: These subprograms return a single value; mainly used to compute and return a value.

A subprogram can be invoked by another subprogram or program which is called the **calling program**.

Procedures

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

- *procedure-name* specifies the name of the procedure.
  - [OR REPLACE] option allows the modification of an existing procedure.
  - The optional parameter list contains name, mode and types of the parameters.
  - *procedure-body* contains the executable part.
  - The AS keyword is used instead of the IS keyword for creating a standalone procedure.

IS, AS: almost synonyms

IN

An IN parameter is ready only. You can reference an IN parameter inside a procedure, but you cannot change its value. Oracle uses IN as the default mode. It means that if you don't specify the mode for a parameter explicitly, Oracle will use the IN mode.

OUT

An OUT parameter is writable. Typically, you set a returned value for the OUT parameter and return it to the calling program. Note that a procedure ignores the value that you supply for an OUT parameter.

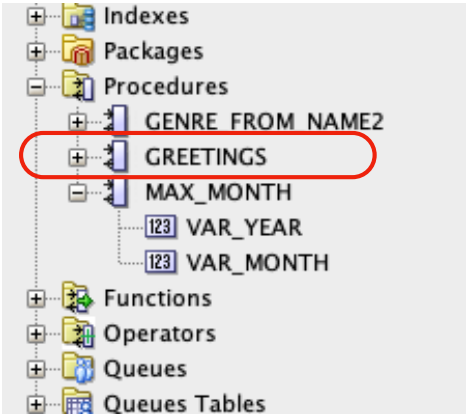
INOUT

An INOUT parameter is both readable and writable. The procedure can read and modify it.

Procedures

Example:

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
```



Executing a Standalone Procedure

A standalone procedure can be called in two ways:

1

```
EXECUTE greetings;
```

2

```
BEGIN
   greetings;
END;
/
```

The procedure can also be called from another PL/SQL block

Deleting a Standalone Procedure:

```
DROP PROCEDURE greetings;
```

Procedures

Here, the procedure takes two numbers using the IN mode and returns their maximum using the OUT parameters.

IN & OUT Mode Example 1:

procedure

block 2

```
set serveroutput on;

DECLARE
   aa number;
   bb number;
   cc number;
PROCEDURE findMax(xx IN number, yy IN number, zz OUT number) IS
BEGIN
   IF xx > yy THEN
      zz:= xx;
   ELSE
      zz:= yy;
   END IF;
END;
BEGIN
   aa:= 618;
   bb:= 421;
   findMax(aa, bb, cc);
   dbms_output.put_line(' Maximum of (618, 421) : ' || cc);
END;
```

Result:

```
Maximum of (618, 421) : 618

PL/SQL procedure successfully completed.
```

Procedures

IN & OUT Mode Example 2:

```
DECLARE
   y number;
PROCEDURE funcNum(x IN OUT number) IS
BEGIN
  x := x * x * x;
END;
BEGIN
   y:= 27;
   funcNum(y);
   dbms_output.put_line(' Outcome of (27): ' || y);
END;
/
```

Result:

```
Outcome of (27): 19683


PL/SQL procedure successfully completed.
```

Functions

A function is same as a procedure except that it returns a value.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
   [ (parameter [,parameter]) ]

   RETURN return_datatype

IS | AS

   [declaration_section]

BEGIN
   executable_section

[EXCEPTION
   exception_section]

END [function_name];
```

The function must contain a return statement.

Functions

Example 1:

```
SELECT * FROM films;
```

FILM_ID	TITLE	YEAR	LANGUAGE	GENRE	DIRECTOR_ID	MAIN_ACTOR_ID	COMPANY_ID
1	Monster, Inc.	01.03.01	English	Animation	5	2	2
2	Kindergarten Cop.	01.03.90	English	Comedy	1	1	1
3	Ratatouille	01.03.07	English	Animation	7	5	2
4	Cars	01.03.06	(null)	Family	3	2	2
5	All about Eve	01.03.50	English	Drama	2	7	3
6	Crash	01.03.04	English	Crime	6	4	4
7	Terminator 2	01.03.91	English	Action	5	1	4
8	Ocean's twelve	01.03.04	English	Crime	9	8	7
9	Avatar	01.03.09	English	Action	5	6	3
10	Solaris	01.03.02	English	Drama	9	8	3
11	Up in the air	01.03.09	(null)	Drama	10	8	8
12	Traffic	01.03.00	English	Crime	9	9	9
13	Maggie	01.03.15	English	Drama	8	1	6

Function:

```
CREATE OR REPLACE FUNCTION totalfilms
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM films;

   RETURN total;
END;
/
```

Calling:

```
DECLARE
   x number(2);
BEGIN
   x := totalfilms();
   dbms_output.put_line('Total no. of Films: ' || x);
END;
/
```

Result:

```
Total no. of Films: 13
```

Functions

Example 2:

function

block 2

```
DECLARE
    aa number;
    bb number;
    cc number;
FUNCTION findMax(xx IN number, yy IN number)
RETURN number
IS
     zz number;
BEGIN
    IF xx > yy THEN
       zz:= xx;
    ELSE
       zz:= yy;
    END IF;
    RETURN zz;
END;
BEGIN
    aa:= 612;
    bb:= 348;
    cc := findMax(aa, bb);
    dbms_output.put_line(' Maximum of 612,348: ' || cc);
END;
/
```

Result 2:

```
Maximum of 612,348: 612


PL/SQL procedure successfully completed.
```

Recursive Functions

When a subprogram calls itself, it is referred to as a recursive call and the process is known as **recursion**.

Factorial function:

```
n! = n*(n-1)!
   = n*(n-1)*(n-2)!
     ...
   = n*(n-1)*(n-2)*(n-3)... 1
```

factorial (5) = 5 * factorial (4)
→ 4 * factorial (3)
→ 3 * factorial (2)
→ 2 * factorial (1)
→ 1 * factorial (0)

Example:

function

block 2

```
DECLARE
   xx number;
   fac number;  --factorial

FUNCTION factorial(xx number)
RETURN number
IS
   ff number;
BEGIN
   IF xx=0 THEN
      ff := 1;
   ELSE
      ff := xx * factorial(xx-1);
   END IF;
   --
RETURN ff;
--
END;


-- new block
BEGIN
   xx:= 11;
   fac  := factorial(xx);
   dbms_output.put_line(' Factorial '|| xx || ' is ' || fac );
END;
/
```

Script output:

```
Factorial 11 is 39916800


PL/SQL procedure successfully completed.
```

Exercise 3

TP3 - Q. 1

1. Ecrire un programme qui interchange les salaires des employées 120 et 122.

```
DECLARE
        - - -
BEGIN
      - - -
    UPDATE - - -
    UPDATE - - -
END;
/
```

TP3 - Q. 2

```
DECLARE

        . . .

BEGIN

        . . .



         IF (n_sal > 10000) THEN

                  . . .

             END IF;
         ELSIF (n_date < DATE '2008-03-19') THEN

                  . . .

             END IF;
         ELSIF (n_sal < 3000) THEN

                  . . .









         . . .



END;
```

2. Ecrire un bloc PL/SQL qui change le pourcentage de commission d'un employé selon les étapes suivante :

a) Afficher une invite pour demander le numéro de l'employé.

b) Après si l'employé :

- a un salaire supérieur à 10000 on lui accorde une commission de 0.4%
- si le salaire est inférieur à 10000 mais il a une expérience de plus de 10 ans on lui accorde une commission de 0.35%
- s'il a un salaire de moins de 3000 on lui accorde une commission de 0.25%
- Dans tous les autres cas on lui donne une commission de 0.15%

TP3 - Q. 3

3. Ecrivez un bloc PL/SQL pour ajouter un nouvel employé:

a. Affichez une invite à saisir le nom de famille d'un nouvel EMPLOYEES ;

b. Affichez une invite à saisir le prénom d'un nouvel EMPLOYEES.

c. Vérifiez que ce nom n'existe pas déjà dans la table EMPLOYEES.

d. Si oui, affichez une phrase comme suit : 'Cet employé existe déjà' et affichez toutes les informations de son job et son salaire actuel ;

```
SET SERVEROUTPUT ON;
DECLARE

        . . .



BEGIN
    FOR emp_id_index IN 1..10 LOOP

            . . .

    IF

          . . .

    ELSE
        INSERT INTO employees (employee_id,last_name,first_name,job_id,manager_id) VALUES (12333, var_lastname, var_firstname, 0001, 0001);
    END IF;
END;
```

TP3 - Q. 4

4. Affichez l'année où il y a eu le plus grand nombre de recrues et le nombre de recrues par mois durant cette année.

```
SET SERVEROUTPUT ON;
DECLARE

      . . .

BEGIN
       . . .

        SELECT hire_date, EXTRACT(YEAR FROM hire_date) AS hired_year FROM employees



           . . .



     FOR index_month IN 1..12
    LOOP

           . . .

        WHERE EXTRACT(YEAR FROM hire_date) = var_year AND EXTRACT (MONTH FROM hire_date) = index_month;

           . . .
    END LOOP;

      . . .

END;
```

Thank you!