7- PROCEDIMENTS I FUNCIONS

Contenido

1. Subprogrames en PL/SQL

Característiques dels subprogrames

- ✓ Un subprograma és un bloc de codi amb **nom**.
- ✓ Els subprogrames són compilats i guardats a la BD.
- ✓ Un cop compilats i validats es poden reutilitzar en vàries aplicacions (*cridant al subprograma*).
- ✓ Per defecte, pertanyen a l'esquema de l'usuari que els ha creat, però es poden compartir amb altres usuaris.
- ✓ Els subprogrames s'executen amb els privilegis del propietari del subprograma, no amb els privilegis de l'usuari que executa el subprograma.

Avantatges dels subprogrames

- ✓ **Millora l'eficiència**: es pot reutilitzar codi ja **compilat** que és a l'àrea de memòria compartida del servidor. No cal tornar a compilar de nou cada cop de l'executem.
- ✓ **Codi compartit** per varis usuaris, per ser a la memòria compartida del servidor.
- ✓ **INTEGRITAT DE LES DADES**: les sentències d'un subprograma, o s'executen totes (**`Statement processed`**) o no se n'executa cap.

Diferències entre subprogrames i blocs anònims

BLOCS ANÒNIMS	SUBPROGRAMES
✓ No tenen nom.	✓ Tenen nom.
✓ Es compilen cada cop que s'executen.	✓ Es compilen un sol cop quan es creen.
✓ No es guarden a la BD.	✓ Es guarden a la BD.
✓ Com no tenen nom, no es poden cridar per altres aplicacions.	✓ Es poden cridar per altres aplicacions.
✓ No retornen valors.	✓ Les funcions poden retornar valors.
✓ No poden tenir paràmetres	✓ Poden tenir paràmetres.

Tipus de subprogrames: procediments i funcions

En PL/SQL hi ha dos tipus de subprogrames:

- ✓ Procediments
- ✓ Funcions

2. Procediments

Característiques

PROCEDIMENTS
✓ Efectuen una acció. ✓ No retornen valors, a no ser que siguin paràmetres de sortida (*OUT o IN OUT parameters*). ✓ No es poden cridar des d'SQL (*suposo que per què no equivalen a un valor*) ✓ No són considerats expressions (*suposo que per què no equivalen a una valor*)

Sintaxi

```
CREATE [OR REPLACE] PROCEDURE nom_proc
     ([parameterName1 [mode1] dataType1,
      parameterName2 [mode2] dataType2,
      ...])
IS|AS
   [local variables, cursors, etc...;]
BEGIN
   SQL and PL/SQL statements;
[EXCEPCION
   WHEN exception_handling actions;
END [nom_proc];
```

OR REPLACE

- ✓ Afegim OR REPLACE si volem sobreescriure la versió actual del procediment.

PARÀMETRES

```
[parameterName1 [mode1] dataType1,  parameterName2 [mode2] dataType2, ...]
```

- ✓ Els paràmetres són **opcionals**.
- ✓ El mode és per defecte **IN** (paràmetres d'entrada)
- ✓ El tipus del paràmetre pot ser IMPLÍCIT (%TYPE) o EXPLÍCIT (VARCHAR2)

CRIDAR UN PROCEDIMENT

Es pot cridar un procediment des:

- ✓ D'un bloc anònim.
- ✓ D'un altre procediment.
- ✓ D'una aplicació.

No es pot cridar a un procediment des d'una sentència SQL.

3. Exemple: hello

Per ser originals, el primer exemple de procediment escriurà '*Hola*'.

Crear el procediment

Per crear el procediment, executar el codi:

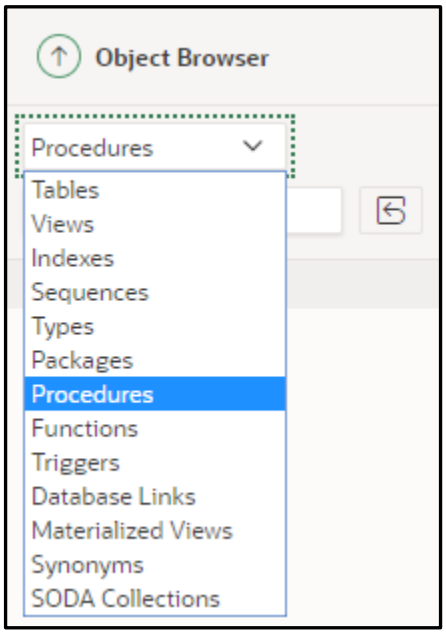
```
CREATE OR REPLACE PROCEDURE hello IS
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello');
END;
```

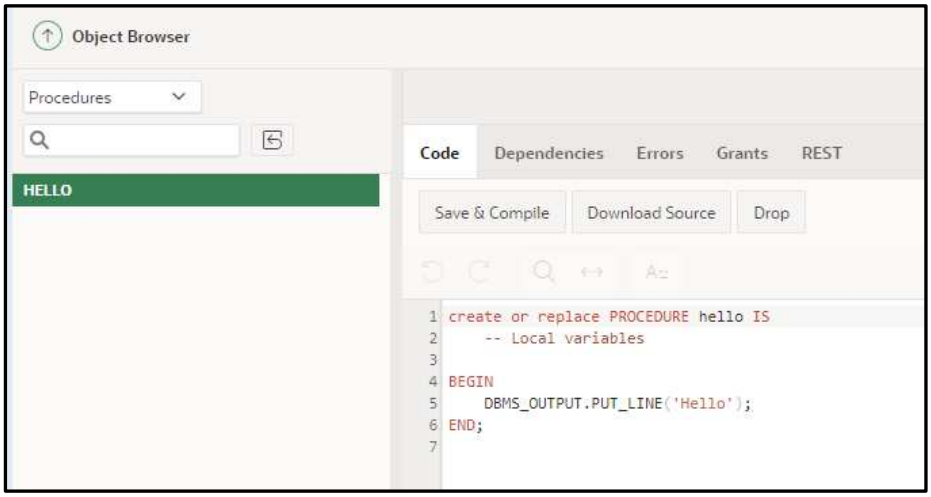
```
Procedure created.
```

NOTA

- ✓ APEX guarda el procediment encara que hi hagin errors.
- ✓ Si hi ha errors, els arreglem i tornem a executar el CREATE. Quan el tornem a executar, com hi **OR REPLACE**, sobreescriu la versió antiga amb errors per la nova.
- ✓ Sinó, hauríem d'esborrar la versió antiga (**DROP**) abans d'executar la nova.

Des de **l'Object Browser d'APEX** es pot veure el procediment que hem creat:



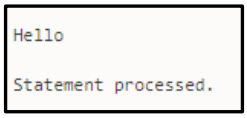


Executar el procediment

Executem el procediment des d'un bloc de codi anònim:

```
BEGIN
   hello;
END;
```

NOTA: Com no té paràmetres, no fa falta posar parèntesi.



4. Exemple: procediment addDepartament

Procediment **addDepartment**, que afegeix un departament a la taula DEPARTMENTS amb:

- ✓ **id**=280
- ✓ **nom=** 'ST-Curriculum'

```
CREATE OR REPLACE PROCEDURE addDepartment IS
    -- Variables locals guarden valors departament nou
    v_dept_id   DEPARTMENTS.DEPARTMENT_ID%TYPE;
    v_dept_name DEPARTMENTS.DEPARTMENT_NAME%TYPE;
BEGIN
    -- Inicialitza les variables amb els valors del nou departament
    v_dept_id   := 280;
    v_dept_name := 'ST-Curriculum ';

    -- Executa INSERT per afegir-lo la taula DEPARTMENTS
    INSERT INTO DEPARTMENTS(DEPARTMENT_ID, DEPARTMENT_NAME)
    VALUES(v_dept_id, v_dept_name);

    -- Mostra quantes columnes ha afagit
   DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || ' rows. ');
END;
```

Executar el procediment:

```
-- Bloc anònim que crida al procediment
BEGIN
     addDepartment;
END;
```

Comprovar que ha afegir el nou departament

```
-- Comprovem que hi ha el nou departament
SELECT * FROM DEPARTMENTS
WHERE DEPARTMENT_ID=280;
```

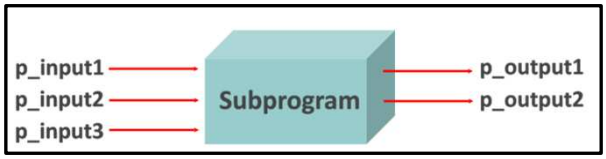
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	ST-Curriculum	-	-

EXERCICIS 1, 2

5. Pas de paràmetres

Tres **tipus** de pas de paràmetres:

- ✓ **IN**: paràmetres d'entrada (per defecte)
- ✓ **OUT**: paràmetres de sortida
- ✓ **IN OUT**: paràmetres d'entrada/sortida



SINTAXI

- ✓ Per conveni, s'acostuma a posar "**p_**" com a prefixe del nom dels paràmetres (igual que en cursors).
- ✓ Quan es defineixen els paràmetres, només s'indica el **tipus de dades**, no la mida.

Paràmetres d'entrada

```
p_student_id IN NUMBER o p_student_id NUMBER
```

- ✓ Utilitzats per enviar valors al procediment.
- ✓ S'indica amb **IN**.
- ✓ Si no s'indica el tipus de paràmetre, per defecte són paràmetres d'entrada (IN).
- ✓ El **valor** del paràmetre **no pot canviar** dintre del procediment. Si es fa una assignació a un paràmetre d'entrada dona un **error** de compilació.

EXEMPLE

Procediment que rep dos valors enters i mostra la suma:

```
CREATE OR REPLACE PROCEDURE suma(p_x IN NUMBER, p_Y IN NUMBER) IS
BEGIN
    -- Mostra la suma dels dos paràmetres
    DBMS_OUTPUT.PUT_LINE(p_x || ' + ' || p_y || ' = ' || (p_x + p_y));
END suma;
```

Crida al procediment:

```
BEGIN
    suma(10, 3);
    suma(5, 13);
END;
```

```
10 + 3 = 13
5 + 13 = 18

Statement processed.
```

Paràmetres de sortida

- ✓ Retornen valors calculats pel procediment a qui el crida.
- ✓ S'indica amb **OUT**

```
p_student_id OUT NUMBER
```

EXEMPLE

Modificació del procediment suma: calcula la suma de dos valors paràmetres enters i la retorna en un tercer paràmetre de sortida.

```
CREATE OR REPLACE PROCEDURE suma(p_x IN NUMBER, p_Y IN NUMBER, p_suma OUT NUMBER) IS
BEGIN
    -- Assigna el 3r paràmetre la suma dels dos primers
    p_suma := p_x + p_y;
END suma;
```

Crida al procediment:

```
DECLARE
    -- Declara la variable per guardar la suma
    v_suma NUMBER;
BEGIN
    suma(10, 3, v_suma);
    DBMS_OUTPUT.PUT_LINE(10 || ' + ' || 3 || ' = ' || v_suma);
    suma(5, 13, v_suma);
    DBMS_OUTPUT.PUT_LINE(5 || ' + ' || 13 || ' = ' || v_suma);
END;
```

Paràmetres d'entrada-sortida

- ✓ S'indica amb **IN OUT.**
- ✓ Actuen com a paràmetres d'entrada (enviar informació al procediment).
- ✓ Actuen com a paràmetres de sortida (retornar valors calculats pel procediment).

```
p_student_id IN OUT NUMBER
```

EXEMPLE

Modificació del procediment suma: calcula la suma i retorna el valor de la suma al segon paràmetre.

```
CREATE OR REPLACE PROCEDURE suma(p_x IN NUMBER, p_y IN OUT NUMBER) IS
BEGIN
    -- Guarda la suma al segon paràmetre
    p_y := p_x + p_y;
END suma;
```

Crida al procediment

```
DECLARE
    v_y NUMBER;
BEGIN
    v_y := 3;
    suma(10, v_y);
    DBMS_OUTPUT.PUT_LINE(10 || ' + ' || 3 || ' = ' || v_y);

    v_y := 13;
    suma(5, v_y);
    DBMS_OUTPUT.PUT_LINE(5 || ' + ' || 13 || ' = ' || v_y);
END;
```

EXEMPLE

Procediment que donat l'id d'un empleat "*retorna*" el nom i el congom.

```
CREATE OR REPLACE PROCEDURE employee_name (p_id IN NUMBER,
                                           p_first_name OUT VARCHAR2,
                                           p_last_name OUT VARCHAR2 ) IS
BEGIN
    -- Busca nom i cognom de l'empleat
    SELECT FIRST_NAME, LAST_NAME
    INTO p_first_name, p_last_name
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = p_id;

    -- Mostra el nom i cognom
    DBMS_OUTPUT.PUT_LINE('PROCEDURE: ' || p_first_name || ' ' ||
                         p_last_name);
END employee_name;
```

Bloc de codi de prova:

```
DECLARE
    -- Declara variables locals per guardar nom i cognom
    v_first_name EMPLOYEES.FIRST_NAME%TYPE;
    v_last_name  EMPLOYEES.LAST_NAME%TYPE;
BEGIN
    -- Crida al procediment amb id 100
    employee_name(100, v_first_name, v_last_name);
    DBMS_OUTPUT.PUT_LINE('NAME: ' || v_first_name || ' ' || v_last_name);

    -- Crida al procediment amb id 144
    employee_name(144, v_first_name, v_last_name);
    DBMS_OUTPUT.PUT_LINE('NAME: ' || v_first_name || ' ' || v_last_name);
END;
```

```
PROCEDURE: Steven KING
NAME: Steven KING
PROCEDURE: Peter VARGAS
NAME: Peter VARGAS

Statement processed.
```

6. Exemple: procediment addDepartament amb paràmetres

Modificar el procediment de l'apartat anterior per a que rebi com a paràmetres l'id i el nom del departament a afegir.

```
CREATE OR REPLACE PROCEDURE addDepartment
     (p_dept_id DEPARTMENTS.DEPARTMENT_ID%TYPE,
      p_dept_name DEPARTMENTS.DEPARTMENT_NAME%TYPE) IS
BEGIN
    INSERT INTO DEPARTMENTS(DEPARTMENT_ID, DEPARTMENT_NAME)
    VALUES(p_dept_id, p_dept_name);
    DBMS_OUTPUT.PUT_LINE('Inserted ' || SQL%ROWCOUNT || ' rows. ');
END;
```

Executar el procediment:

```
BEGIN
    addDepartment (1000, 'Pepes department');
    addDepartment (1001, '1001 department');
    addDepartment (1002, 'LAST department');
END;
```

```
Inserted 1 rows.
Inserted 1 rows.
Inserted 1 rows.

Statement processed.
```

Comprovar el resultat:

```
SELECT *
FROM DEPARTMENTS
WHERE DEPARTMENT_ID>=1000
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1000	Pepes department	-	-
1001	1001 department	-	-
1002	LAST department	-	-

3 rows returned in 0.00 seconds Download

7. Valors per defecte

Quan es crida a un procediment, tots els paràmetres **IN** del procediment han de tenir valors, sinó APEX mostra un error:

```
add_dept ('EDUCATION');

ORA-06550:  line 2, column 1:
PLS-00306:  wrong number or types of arguments in call to
    'ADD_DEPT'
ORA-06550:  line 2, column 1:
PL/SQL:  Statement ignored
1.  begin
2.  add_dept('EDUCATION');
3.  end;
```

Es pot assignar un **valor per defecte** a un paràmetre **IN** en la declaració del procediment: si el paràmetre no rep valor quan es crida al procediment se li assigna el valor per defecte.

```
CREATE OR REPLACE PROCEDURE default_values_proc
     (p_name VARCHAR := 'Unknown',p_loc NUMBER DEFAULT 1400) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('p_name:' || p_name || ' p_loc: ' || p_loc);
END;
```

Es pot cridar al procediment:

```
BEGIN
   default_values_proc;
END;
```

Crida sense paràmetres. S'assigna als dos paràmetres el valor per defecte:

```
p_name:Unknown p_loc: 1400
```

```
BEGIN
   default_values_proc('education');
END;
```

Només es dona valor al primer paràmetre. El segon té el valor per defecte:

```
p_name:education p_loc: 1400
```

```
BEGIN
    default_values_proc('education', 1000);
END;
```

Els dos paràmetres tenen valor en la crida:

```
p_name:education p_loc: 1000
```

EXERCICIS 3, 4, 5

8. Funcions

FUNCIONS

- ✓ Retornen un valor. A més, poden tenir paràmetres de sortida (*OUT o IN OUT parameters*).
- ✓ Es poden cridar des d'SQL (*seran substituïdes pel valor que retornen*).
- ✓ Són considerades expressions (*seran substituïdes pel valor que retornen*).

SINTAXI

```
CREATE OR REPLACE FUNCTION <nomfunció> [(<llista de paràmetres>)]
RETURN <tipus_valor_retornat> IS
   <declaracions>;
BEGIN
     <instruccions>;
RETURN <expressió>;
...
EXCEPTION
     <excepcions>;
END [<nomfunció>];
```

Crida a la funció:

```
<variable> := <nomfunció>(paràmetres);
```

EXEMPLE

Funció que rep dos paràmetres enters i retorna la seva suma

```
CREATE OR REPLACE FUNCTION suma(p_x IN NUMBER, p_Y IN NUMBER)
RETURN NUMBER IS
BEGIN
    -- Retorna la suma dels dos valors
    return p_x + p_y;
END suma;
```

Crida a la funció:

```
DECLARE
    v_suma NUMBER;
BEGIN
    v_suma := suma(10, 3);
    DBMS_OUTPUT.PUT_LINE(10 || ' + ' || 3 || ' = ' || v_suma);
    DBMS_OUTPUT.PUT_LINE(5 || ' + ' || 13 || ' = ' || suma(5, 13));
END;
```

9. NO_DATA_FOUND exception

Com les funcions sempre retornen un valor, és molt important tractar l'excepció **NO_DATA_FOUND**.

EXEMPLE

- ✓ Crear la funció **get_salary** que, donat l'id d'un empleat, retorni el seu salari.
- ✓ Fer un bloc de codi que provi la funció amb els ids 100, 200 i 300.

DECLARACIÓ DE LA FUNCIÓ

```
CREATE OR REPLACE FUNCTION get_salary
   (p_employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
   RETURN NUMBER IS
       -- Declara la variable local v_salary
      v_salary EMPLOYEES.SALARY%TYPE;
BEGIN
    -- Busca el salari de l'empleat
    SELECT SALARY
    INTO v_salary
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = p_employee_id;

    -- Retorna el nombre salari
    RETURN v_salary ;
END;
```

PROVA DE LA FUNCIÓ AMB id=100

```
BEGIN
     DBMS_OUTPUT.PUT_LINE(get_salary(100));
END;
```

```
24000

Statement processed.
```

PROVA DE LA FUNCIÓ AMB id=300

```
BEGIN
      DBMS_OUTPUT.PUT_LINE(get_salary(300));
END;
```

Com no hi ha cap empleat amb id 300 → mostra missatge d'error.

```
ORA-01403: no data found
ORA-06512: at "DMARTIEMPLOYEES.GET_SALARY", line 8
ORA-06512: at line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Capturem l'excepció **NO_DATA_FOUND** i retornem valor 0 en el cas en que no existeix l'empleat:

```
CREATE OR REPLACE FUNCTION get_salary
   (p_employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
   RETURN NUMBER IS
       -- Declara la variable local v_salary
       v_salary EMPLOYEES.SALARY%TYPE;
   BEGIN
       -- Busca el salari de l'empleat
       SELECT SALARY
       INTO v_salary
       FROM EMPLOYEES
       WHERE EMPLOYEE_ID = p_employee_id;

      -- Retorna el nombre salari
       RETURN v_salary ;
   EXCEPTION
       WHEN NO_DATA_FOUND THEN
           -- Si el treballador no existeix, retorna salary 0
           RETURN 0;
   END;
```

Tornem a fer la prova:

```
DECLARE
      BEGIN
      DBMS_OUTPUT.PUT_LINE(get_salary(300));
END;
```

```
0

Statement processed.
```

EXERCICIS 6, 7, 8