

## PL/SQL - Ejercicios de refuerzo

**Nota previa:** Todos los scripts deben ir acompañados de un bloque anónimo y del correspondiente tratamiento de excepciones.

### 1. Crea un procedimiento que actualice el ganador del premio de un puerto.

```
CREATE OR REPLACE PROCEDURE p_actualizarGanador(V_PUERTO IN PUERTO.NOMPUERTO%TYPE,
V_DORSAL IN PUERTO.DORSAL%TYPE)
```

```
IS
  NO_EXISTE_PUERTO EXCEPTION;
```

```
BEGIN
  UPDATE PUERTO
  SET DORSAL = V_DORSAL
  WHERE NOMPUERTO = V_PUERTO;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE NO_EXISTE_PUERTO;
  END IF;
```

```
EXCEPTION
  WHEN NO_EXISTE_PUERTO THEN
    DBMS_OUTPUT.PUT_LINE('No existe ese puerto');
```

```
END p_actualizarGanador;
```

```
-- bloque anónimo
```

```
DECLARE
  V_PUERTO1 PUERTO.NOMPUERTO%TYPE;
  V_DORSAL1 PUERTO.DORSAL%TYPE;
```

```
BEGIN
  V_PUERTO1 := '&PUERTO';
  V_DORSAL1 := '&DORSAL';

  p_actualizarGanador(V_PUERTO1, V_DORSAL1);
```

```
END;
```

```
-- 1. Crea un procedimiento que actualice el ganador del premio de un puerto.
CREATE OR REPLACE PROCEDURE p_actualizarGanador(V_PUERTO IN PUERTO.NOMPUERTO%TYPE, V_DORSAL IN PUERTO.DORSAL%TYPE)
IS
  NO_EXISTE_PUERTO EXCEPTION;
BEGIN
  UPDATE PUERTO
  SET DORSAL = V_DORSAL
  WHERE NOMPUERTO = V_PUERTO;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE NO_EXISTE_PUERTO;
  END IF;

EXCEPTION
  WHEN NO_EXISTE_PUERTO THEN
    DBMS_OUTPUT.PUT_LINE('No existe ese puerto');
END p_actualizarGanador;
```

Salida de Script x  
Tarea terminada en 0,106 segundos  
Procedure P\_ACTUALIZARGANADOR compilado

```
-- bloque anónimo
DECLARE
  V_PUERTO1 PUERTO.NOMPUERTO%TYPE;
  V_DORSAL1 PUERTO.DORSAL%TYPE;

BEGIN
  V_PUERTO1 := '&PUERTO';
  V_DORSAL1 := '&DORSAL';

  p_actualizarGanador(V_PUERTO1, V_DORSAL1);
END;
```

Salida de Script x  
Tarea terminada en 6,068 segundos  
Antiguo:DECLARE  
V\_PUERTO1 PUERTO.NOMPUERTO%TYPE;  
V\_DORSAL1 PUERTO.DORSAL%TYPE;  
BEGIN  
V\_PUERTO1 := '&PUERTO';  
V\_DORSAL1 := '&DORSAL';  
  
p\_actualizarGanador(V\_PUERTO1, V\_DORSAL1);  
END;  
Nuevo:DECLARE  
V\_PUERTO1 PUERTO.NOMPUERTO%TYPE;  
V\_DORSAL1 PUERTO.DORSAL%TYPE;  
BEGIN  
V\_PUERTO1 := 'Sierra Nevada';  
V\_DORSAL1 := '99';  
  
p\_actualizarGanador(V\_PUERTO1, V\_DORSAL1);  
END;  
Procedimiento PL/SQL terminado correctamente.

	NOMPUERTO	ALTURA	CATEGORIA	PENDIENTE	NETAPA	DORSAL
1	Alto del Naranco	5651		6,9	1030	
2	Arcalis	2230	E	6,5	1090	
3	Cerler-Circo de Ampriu	2500	E	5,87	119	
4	Coll de la Comella	1362	1	8,07	102	
5	Coll de Ordino	1980	E	5,3	107	
6	Cruz de la Demanda	1850	E	7	1120	
7	Lagos de Covadonga	1134	E	6,86	1642	
8	Navacerrada	1860	1	7,5	192	
9	Puerto de Alisas	672	1	5,8	151	
10	Puerto de la Morcuera	1760	2	6,5	192	
11	Puerto de Mijares	1525	1	4,9	1824	
12	Puerto de Navalmoral	1521	2	4,3	182	
13	Puerto de Pedro Bernardo	1250	1	4,2	1825	
14	Sierra Nevada	2300	E	6	99	

2. Crea un procedimiento que enumere los puertos con una pendiente mayor que 6. Utiliza ambos tipos de cursor explícito.

```
CREATE OR REPLACE PROCEDURE p_enumerarPuertos
```

```
IS
```

```
V_PUERTOS PUERTO%ROWTYPE;
```

```
CURSOR c_datos1 IS
```

```
SELECT *
FROM PUERTO
WHERE PENDIENTE > 6;
```

```
BEGIN
```

```
OPEN c_datos1;
```

```
FETCH c_datos1 INTO V_PUERTOS;
```

```
WHILE c_datos1%FOUND LOOP
```

```
DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' ||
```

```
V_PUERTOS.NOMPUERTO);
```

```
DBMS_OUTPUT.PUT_LINE('Altura puerto: ' ||
```

```
V_PUERTOS.ALTURA);
```

```
DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' ||
```

```
V_PUERTOS.CATEGORIA);
```

```
DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' ||
```

```
V_PUERTOS.PENDIENTE);
```

```
DBMS_OUTPUT.PUT_LINE('Numero etapa: ' ||
```

```
V_PUERTOS.NETAPA);
```

```
DBMS_OUTPUT.PUT_LINE('Dorsal' ||
```

```
V_PUERTOS.DORSAL);
```

```
FETCH c_datos1 INTO V_PUERTOS;
```

```
END LOOP;
```

```
CLOSE c_datos1;
```

```
END p_enumerarPuertos;
```

```
-- bloque anónimo
```

```
DECLARE
```

```
BEGIN
```

```
p_enumerarPuertos;
```

```
END;
```

```
-- 2. Crea un procedimiento que enumere los puertos con una pendiente mayor que 6.

CREATE OR REPLACE PROCEDURE p_enumerarPuertos

IS
    V_PUERTOS PUERTO%ROWTYPE;

    CURSOR c_datos1 IS
        SELECT *
        FROM PUERTO
        WHERE PENDIENTE > 6;

BEGIN
    OPEN c_datos1;
    FETCH c_datos1 INTO V_PUERTOS;
    WHILE c_datos1%FOUND LOOP
        DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' || V_PUERTOS.NOMPUERTO);
        DBMS_OUTPUT.PUT_LINE('Altura puerto: ' || V_PUERTOS.ALTURA);
        DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' || V_PUERTOS.CATEGORIA);
        DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' || V_PUERTOS.PENDIENTE);
        DBMS_OUTPUT.PUT_LINE('Numero etapa: ' || V_PUERTOS.NETAPA);
        DBMS_OUTPUT.PUT_LINE('Dorsal' || V_PUERTOS.DORSAL);
        FETCH c_datos1 INTO V_PUERTOS;
    END LOOP;
    CLOSE c_datos1;
END p_enumerarPuertos;
```

```
Nombre puerto: Alto del Naranco
Altura puerto: 565
Categoria puerto: 1
Pendiente puerto: 6,9
Numero etapa: 10
Dorsal30
Nombre puerto: Arcalis
Altura puerto: 2230
Categoria puerto: E
Pendiente puerto: 6,5
Numero etapa: 10
Dorsal90
Nombre puerto: Coll de la Comella
Altura puerto: 1362
Categoria puerto: 1
Pendiente puerto: 8,07
Numero etapa: 10
Dorsal2
Nombre puerto: Cruz de la Demanda
Altura puerto: 1850
Categoria puerto: E
Pendiente puerto: 7
Numero etapa: 11
Dorsal20
Nombre puerto: Lagos de Covadonga
Altura puerto: 1134
Categoria puerto: E
Pendiente puerto: 6,86
Numero etapa: 16
Dorsal42
Nombre puerto: Navacerrada
Altura puerto: 1860
Categoria puerto: 1
Pendiente puerto: 7,5
Numero etapa: 19
Dorsal2
Nombre puerto: Puerto de la Morcuera
Altura puerto: 1760
Categoria puerto: 2
Pendiente puerto: 6,5
Numero etapa: 19
Dorsal2
```

3. Crea una función que acepte el nombre del puerto y devuelva la pendiente del mismo. Si la pendiente es superior a 7, debe lanzarse una excepción de usuario.

```
CREATE OR REPLACE FUNCTION f_devolverPendientePuerto(V_NOM_PUERTO IN
PUERTO.NOMPUERTO%TYPE)
RETURN PUERTO.PENDIENTE%TYPE
```

```
IS
  V_PENDIENTE PUERTO.PENDIENTE%TYPE;
  PENDIENTE_MAYOR EXCEPTION;
```

```
BEGIN
  SELECT PENDIENTE INTO V_PENDIENTE
  FROM PUERTO
  WHERE NOMPUERTO = V_NOM_PUERTO;
```

```
IF V_PENDIENTE > 7 THEN
  RAISE PENDIENTE_MAYOR;
END IF;
```

```
RETURN V_PENDIENTE;
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Este puerto no existe');
  WHEN PENDIENTE_MAYOR THEN
    DBMS_OUTPUT.PUT_LINE('Este puerto tiene una pendiente mayor que 7');
```

```
END;
```

```
-- bloque anónimo
```

```
DECLARE
  V_NOMBRE_PUERTO PUERTO.NOMPUERTO%TYPE;
  V_PENDIENTE1 PUERTO.PENDIENTE%TYPE;
```

```
BEGIN
  V_NOMBRE_PUERTO := '&NOMPUERTO';

  V_PENDIENTE1 := f_devolverPendientePuerto(V_NOMBRE_PUERTO);
```

```
DBMS_OUTPUT.PUT_LINE('El puerto es: ' || V_NOMBRE_PUERTO);
DBMS_OUTPUT.PUT_LINE('y su pendiente es: ' || V_PENDIENTE1);
```

```
END;
```

```
-- 3. Crea una función que acepte el nombre del puerto y devuelva la pendiente del mismo.
-- Si la pendiente es superior a 7, debe lanzarse una excepción de usuario.

CREATE OR REPLACE FUNCTION f_devolverPendientePuerto(V_NOM_PUERTO IN PUERTO.NOMPUERTO%TYPE)
RETURN PUERTO.PENDIENTE%TYPE

IS
  V_PENDIENTE PUERTO.PENDIENTE%TYPE;
  PENDIENTE_MAYOR EXCEPTION;

BEGIN
  SELECT PENDIENTE INTO V_PENDIENTE
  FROM PUERTO
  WHERE NOMPUERTO = V_NOM_PUERTO;

  IF V_PENDIENTE > 7 THEN
    RAISE PENDIENTE_MAYOR;
  END IF;

  RETURN V_PENDIENTE;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Este puerto no existe');
  WHEN PENDIENTE_MAYOR THEN
    DBMS_OUTPUT.PUT_LINE('Este puerto tiene una pendiente mayor que 7');

END;
```

```
-- bloque anónimo

DECLARE
  V_NOMBRE_PUERTO PUERTO.NOMPUERTO%TYPE;
  V_PENDIENTE1 PUERTO.PENDIENTE%TYPE;

BEGIN
  V_NOMBRE_PUERTO := '&NOMPUERTO';

  V_PENDIENTE1 := f_devolverPendientePuerto(V_NOMBRE_PUERTO);

  DBMS_OUTPUT.PUT_LINE('El puerto es: ' || V_NOMBRE_PUERTO);
  DBMS_OUTPUT.PUT_LINE('y su pendiente es: ' || V_PENDIENTE1);

END;
```

```
Este puerto tiene una pendiente mayor que 7
```

*Ejemplo con "Navacerrada"*

```
El puerto es: Sierra Nevada
y su pendiente es: 6
```

4. Crea un procedimiento que acepte una categoría de puerto y liste todos los puertos de esa categoría. Utiliza ambos tipos de cursor explícito.

```
CREATE OR REPLACE PROCEDURE p_listarPuertosCategoria(V_CATEGORIA IN PUERTO.CATEGORIA%TYPE)
```

```
IS
```

```
V_PUERTOS PUERTO%ROWTYPE;
```

```
CURSOR c_datos1 IS
```

```
SELECT * INTO V_PUERTOS
```

```
FROM PUERTO
```

```
WHERE CATEGORIA = V_CATEGORIA;
```

```
BEGIN
```

```
OPEN c_datos1;
```

```
FETCH c_datos1 INTO V_PUERTOS;
```

```
WHILE c_datos1%FOUND LOOP
```

```
DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' ||
```

```
V_PUERTOS.NOMPUERTO);
```

```
DBMS_OUTPUT.PUT_LINE('Altura puerto: ' ||
```

```
V_PUERTOS.ALTURA);
```

```
DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' ||
```

```
V_PUERTOS.CATEGORIA);
```

```
DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' ||
```

```
V_PUERTOS.PENDIENTE);
```

```
DBMS_OUTPUT.PUT_LINE('Numero etapa: ' ||
```

```
V_PUERTOS.NETAPA);
```

```
DBMS_OUTPUT.PUT_LINE('Dorsal' ||
```

```
V_PUERTOS.DORSAL);
```

```
FETCH c_datos1 INTO V_PUERTOS;
```

```
END LOOP;
```

```
CLOSE c_datos1;
```

```
END p_listarPuertosCategoria;
```

```
-- bloque anónimo
```

```
DECLARE
```

```
V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;
```

```
BEGIN
```

```
V_CATEGORIA1 := '&CATEGORIA';
```

```
p_listarPuertosCategoria(V_CATEGORIA1);
```

```
END p_listarPuertosCategoria;
```

```
-- 4. Crea un procedimiento que acepte una categoría de puerto y liste todos los puertos de esa categoría. Utiliza ambos tipos de cursor explícito.

CREATE OR REPLACE PROCEDURE p_listarPuertosCategoria(V_CATEGORIA IN PUERTO.CATEGORIA%TYPE)

IS

    V_PUERTOS PUERTO%ROWTYPE;

    CURSOR c_datos1 IS
        SELECT * INTO V_PUERTOS
        FROM PUERTO
        WHERE CATEGORIA = V_CATEGORIA;

BEGIN

    OPEN c_datos1;
    FETCH c_datos1 INTO V_PUERTOS;
    WHILE c_datos1%FOUND LOOP
        DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' || V_PUERTOS.NOMPUERTO);
        DBMS_OUTPUT.PUT_LINE('Altura puerto: ' || V_PUERTOS.ALTURA);
        DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' || V_PUERTOS.CATEGORIA);
        DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' || V_PUERTOS.PENDIENTE);
        DBMS_OUTPUT.PUT_LINE('Numero etapa: ' || V_PUERTOS.NETAPA);
        DBMS_OUTPUT.PUT_LINE('Dorsal' || V_PUERTOS.DORSAL);
        FETCH c_datos1 INTO V_PUERTOS;
    END LOOP;
    CLOSE c_datos1;

END p_listarPuertosCategoria;
```

Salida de Script: Tarea terminada en 0,106 segundos

Procedure P\_LISTARPUERTOSCATEGORIA compilado

```
-- bloque anónimo

DECLARE
    V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;

BEGIN
    V_CATEGORIA1 := '&CATEGORIA';

    p_listarPuertosCategoria(V_CATEGORIA1);

END p_listarPuertosCategoria;

Nuevo:DECLARE
    V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;

BEGIN
    V_CATEGORIA1 := '1';

    p_listarPuertosCategoria(V_CATEGORIA1);

END p_listarPuertosCategoria;

Procedimiento PL/SQL terminado correctamente.
```

Salida de Script: Tarea terminada en 1,7 segundos

Antiguo:DECLARE  
V\_CATEGORIA1 PUERTO.CATEGORIA%TYPE;  
BEGIN  
V\_CATEGORIA1 := '&CATEGORIA';  
p\_listarPuertosCategoria(V\_CATEGORIA1);  
END p\_listarPuertosCategoria;  
Nuevo:DECLARE  
V\_CATEGORIA1 PUERTO.CATEGORIA%TYPE;  
BEGIN  
V\_CATEGORIA1 := '1';  
p\_listarPuertosCategoria(V\_CATEGORIA1);  
END p\_listarPuertosCategoria;  
Procedimiento PL/SQL terminado correctamente.

```
Ej3_CICLISMO12 x
Nombre puerto: Alto del Naranco
Altura puerto: 565
Categoria puerto: 1
Pendiente puerto: 6,9
Numero etapa: 10
Dorsal30
Nombre puerto: Coll de la Comella
Altura puerto: 1362
Categoria puerto: 1
Pendiente puerto: 8,07
Numero etapa: 10
Dorsal2
Nombre puerto: Navacerrada
Altura puerto: 1860
Categoria puerto: 1
Pendiente puerto: 7,5
Numero etapa: 19
Dorsal2
Nombre puerto: Puerto de Alisas
Altura puerto: 672
Categoria puerto: 1
Pendiente puerto: 5,8
Numero etapa: 15
Dorsal1
Nombre puerto: Puerto de Mijares
Altura puerto: 1525
Categoria puerto: 1
Pendiente puerto: 4,9
Numero etapa: 18
Dorsal24
Nombre puerto: Puerto de Pedro Bernardo
Altura puerto: 1250
Categoria puerto: 1
Pendiente puerto: 4,2
Numero etapa: 18
Dorsal25
```

5. Crea una función que acepte un número de dorsal y retorne la cantidad de puertos que ha ganado. Si ha ganado más de 1 puerto, lanzar la excepción “experto”.

```
CREATE OR REPLACE FUNCTION f_cantidadPuertosGanados (V_DORSAL IN PUERTO.DORSAL%TYPE)
RETURN NUMBER
```

```
IS
  V_VICTORIAS NUMBER;
  EXPERTO EXCEPTION;

BEGIN
  SELECT COUNT(*) INTO V_VICTORIAS
  FROM PUERTO
  WHERE DORSAL = V_DORSAL;

  RETURN V_VICTORIAS;

  IF V_VICTORIAS > 1 THEN
    RAISE EXPERTO;
  END IF;

EXCEPTION
  WHEN EXPERTO THEN
    DBMS_OUTPUT.PUT_LINE('ha ganado más de 1 puerto, es
nivel experto');

END;
```

```
-- bloque anónimo
```

```
DECLARE
  V_DORSAL PUERTO.DORSAL%TYPE;
  V_NUM_VICTORIAS NUMBER;

BEGIN
  V_DORSAL := &DORSAL;

  V_NUM_VICTORIAS := f_cantidadPuertosGanados(V_DORSAL);

  DBMS_OUTPUT.PUT_LINE(V_NUM_VICTORIAS);

END;
```

```
-- 5. Crea una función que acepte un número de dorsal y retorne la cantidad de puertos que ha ganado. Si ha ganado más de 1 puerto, lanzar la excepción "experto".

CREATE OR REPLACE FUNCTION f_cantidadPuertosGanados (V_DORSAL IN PUERTO.DORSAL%TYPE)
RETURN NUMBER
IS
  V_VICTORIAS NUMBER;
  EXPERTO EXCEPTION;

BEGIN
  SELECT COUNT(*) INTO V_VICTORIAS
  FROM PUERTO
  WHERE DORSAL = V_DORSAL;

  RETURN V_VICTORIAS;

  IF V_VICTORIAS > 1 THEN
    RAISE EXPERTO;
  END IF;

EXCEPTION
  WHEN EXPERTO THEN
    DBMS_OUTPUT.PUT_LINE('ha ganado más de 1 puerto, es nivel experto');

END;
```

Function F\_CANTIDADPUERTOSGANADOS compilado

```
DECLARE
  V_DORSAL PUERTO.DORSAL%TYPE;
  V_NUM_VICTORIAS NUMBER;

BEGIN
  V_DORSAL := &DORSAL;

  V_NUM_VICTORIAS := f_cantidadPuertosGanados(V_DORSAL);

  DBMS_OUTPUT.PUT_LINE(V_NUM_VICTORIAS);

END;
```

Antiguo:DECLARE  
V\_DORSAL PUERTO.DORSAL%TYPE;  
V\_NUM\_VICTORIAS NUMBER;  
  
BEGIN  
V\_DORSAL := &DORSAL;  
  
V\_NUM\_VICTORIAS := f\_cantidadPuertosGanados(V\_DORSAL);  
  
DBMS\_OUTPUT.PUT\_LINE(V\_NUM\_VICTORIAS);  
  
END;  
Nuevo:DECLARE  
V\_DORSAL PUERTO.DORSAL%TYPE;  
V\_NUM\_VICTORIAS NUMBER;  
  
BEGIN  
V\_DORSAL := 2;  
  
V\_NUM\_VICTORIAS := f\_cantidadPuertosGanados(V\_DORSAL);  
  
DBMS\_OUTPUT.PUT\_LINE(V\_NUM\_VICTORIAS);  
  
END;  
Procedimiento PL/SQL terminado correctamente.

6. Crea un procedimiento que acepte un número de etapa y una cantidad de kilómetros y actualice los datos de la etapa. Además debe devolver (dato de salida) el dorsal del ciclista que la ganó.

```
CREATE OR REPLACE PROCEDURE p_actualizarDatosEtapa (V_ETAPA IN ETAPA.NETAPA%TYPE,
V_KILOMETROS IN ETAPA.KM%TYPE, V_DORSAL OUT ETAPA.DORSAL%TYPE)
```

```
IS
```

```
BEGIN
```

```
    UPDATE ETAPA
    SET KM = V_KILOMETROS
    WHERE NETAPA = V_ETAPA;
```

```
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay etapas
con este numero');
```

```
END p_actualizarDatosEtapa;
```

```
-- bloque anónimo
```

```
DECLARE
```

```
    V_ETAPA ETAPA.NETAPA%TYPE;
    V_KILOMETROS ETAPA.KM%TYPE;
    V_DORSAL ETAPA.DORSAL%TYPE;
```

```
BEGIN
```

```
    V_ETAPA := '&ETAPA';
    V_KILOMETROS := '&KILOMETROS';
```

```
    p_actualizarDatosEtapa(V_ETAPA, V_KILOMETROS, V_DORSAL);
```

```
    DBMS_OUTPUT.PUT_LINE(V_DORSAL);
```

```
END;
```

```
-- 6. Crea un procedimiento que acepte un número de etapa y una cantidad de kilómetros y actualice los datos de la etapa.
-- Además debe devolver (dato de salida) el dorsal del ciclista que la ganó.

CREATE OR REPLACE PROCEDURE p_actualizarDatosEtapa (V_ETAPA IN ETAPA.NETAPA%TYPE, V_KILOMETROS IN ETAPA.KM%TYPE, V_DORSAL OUT ETAPA.DORSAL%TYPE)
IS
BEGIN
    UPDATE ETAPA
    SET KM = V_KILOMETROS
    WHERE NETAPA = V_ETAPA;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay etapas con este numero');

END p_actualizarDatosEtapa;
```

Salida de Script: Tarea terminada en 0,154 segundos  
Procedure P\_ACTUALIZADATOSSETAPA compilado

```
-- bloque anónimo

DECLARE
    V_ETAPA ETAPA.NETAPA%TYPE;
    V_KILOMETROS ETAPA.KM%TYPE;
    V_DORSAL ETAPA.DORSAL%TYPE;

BEGIN
    V_ETAPA := '&ETAPA';
    V_KILOMETROS := '&KILOMETROS';

    p_actualizarDatosEtapa(V_ETAPA, V_KILOMETROS, V_DORSAL); -- le he puesto a la NETAPA 2 unos 10KM
    DBMS_OUTPUT.PUT_LINE(V_DORSAL);

END;
```

Salida de Script: Tarea terminada en 7,541 segundos

Antiguo: DECLARE  
V\_ETAPA ETAPA.NETAPA%TYPE;  
V\_KILOMETROS ETAPA.KM%TYPE;  
V\_DORSAL ETAPA.DORSAL%TYPE;

BEGIN  
V\_ETAPA := '&ETAPA';  
V\_KILOMETROS := '&KILOMETROS';

p\_actualizarDatosEtapa(V\_ETAPA, V\_KILOMETROS, V\_DORSAL); -- le he puesto a la NETAPA 2 unos 10KM

DBMS\_OUTPUT.PUT\_LINE(V\_DORSAL);

END;

Nuevo: DECLARE  
V\_ETAPA ETAPA.NETAPA%TYPE;  
V\_KILOMETROS ETAPA.KM%TYPE;  
V\_DORSAL ETAPA.DORSAL%TYPE;

BEGIN  
V\_ETAPA := '1';  
V\_KILOMETROS := '10';

p\_actualizarDatosEtapa(V\_ETAPA, V\_KILOMETROS, V\_DORSAL); -- le he puesto a la NETAPA 2 unos 10KM

DBMS\_OUTPUT.PUT\_LINE(V\_DORSAL);

END;

Procedimiento PL/SQL terminado correctamente.

	NETAPA	KM	SALIDA	LLEGADA	DORSAL
1	1	10	Valladolid	Valladolid	1
2	2	10	Valladolid	Salamanca	36
3	3	240	Salamanca	Caceres	12

7. Crea un procedimiento que acepte un nombre de equipo y un nombre de director y añada éste a la tabla de equipos.

```
CREATE OR REPLACE PROCEDURE p_añadirDirector(V_EQUIPO IN EQUIPO.NOMEQ%TYPE, V_DIRECTOR IN EQUIPO.DESCRIPCION%TYPE)
```

```
IS
```

```
BEGIN
```

```
  INSERT INTO EQUIPO (NOMEQ, DESCRIPCION)
  VALUES (V_EQUIPO, V_DIRECTOR);
```

```
END p_añadirDirector;
```

```
-- bloque anónimo
```

```
DECLARE
```

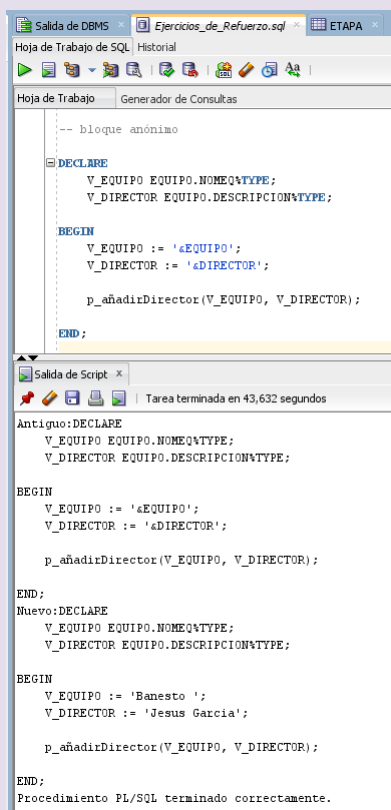
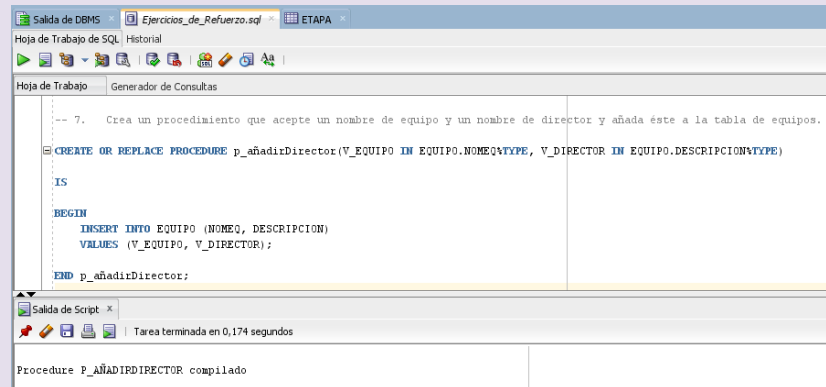
```
  V_EQUIPO EQUIPO.NOMEQ%TYPE;
  V_DIRECTOR EQUIPO.DESCRIPCION%TYPE;
```

```
BEGIN
```

```
  V_EQUIPO := '&EQUIPO';
  V_DIRECTOR := '&DIRECTOR';
```

```
  p_añadirDirector(V_EQUIPO, V_DIRECTOR);
```

```
END;
```



NOMEQ	DESCRIPCION
1 ValleInclan	Fran Muñoz
2 Amore Vita	Ricardo Padacci
3 Artiach	Josão Perez
4 Banesto	Miguel Echevarria
5 Bresciali-Refin	Pietro Armani
6 Carrera	Luigi Petroni
7 Castorama	Jean Philip
8 Euskadi	Pedro Txucaru
9 Gatorade	Gian Luca Paccelli
10 Gewiss	Moreno Argentin
11 Jolly Club	Johan Richard
12 Kelme	Adlvaro Pino
13 Lotus Festina	Suarez Cuevas
14 Mapei-Clas	Juan Fernandez
15 Mercatone Uno	Ettore Romano
16 Motorola	John Fidwell
17 Navigare	Lonrenzo Sciacchi
18 ONCE	Manuel Sainz
19 PDM	Piet Van Der Kruis
20 Seguros Amaya	Minguez
21 Telecom	Morgan Reikcard
22 TVM	Steveens Henk
23 Wordperfect	Bill Gates
24 Banesto	Jesus Garcia



8. Crea una función que acepte un nombre de equipo y devuelva la cantidad de integrantes del equipo. En el caso de haber menos de 5 ciclistas, lanzar una excepción de usuario sin nombre que será recogida en el bloque anónimo.

```
CREATE OR REPLACE FUNCTION f_cantidadCiclistasEquipo (V_EQUIPO IN CICLISTA.NOMEQ%TYPE)
RETURN NUMBER
```

```
IS
  V_NUM_CICLISTAS NUMBER;

BEGIN

  SELECT COUNT(*) INTO V_NUM_CICLISTAS
  FROM CICLISTA
  WHERE NOMEQ = V_EQUIPO;

  IF V_NUM_CICLISTAS < 5 THEN
    RAISE_APPLICATION_ERROR (-20006, 'hay
menos de 5 ciclistas en este equipo, son muy pocos...');
  END IF;

  RETURN V_NUM_CICLISTAS;

END;
```

-- bloque anónimo

```
DECLARE
  V_NOM_EQUIPO CICLISTA.NOMEQ%TYPE;
  V_NUM_CICLISTAS NUMBER;

  EQUIPO_PEQUEÑO EXCEPTION;
  PRAGMA EXCEPTION_INIT(EQUIPO_PEQUEÑO, -20006);

BEGIN
  V_NOM_EQUIPO := '&EQUIPO';

  V_NUM_CICLISTAS := f_cantidadCiclistasEquipo(V_NOM_EQUIPO);

  DBMS_OUTPUT.PUT_LINE(V_NUM_CICLISTAS);

EXCEPTION
  WHEN EQUIPO_PEQUEÑO THEN
    DBMS_OUTPUT.PUT_LINE('hay menos de 5 ciclistas en este equipo, son muy
pocos...');

END;
```

```
-- 8. Crea una función que acepte un nombre de equipo y devuelva la cantidad de integrantes del equipo.
-- En el caso de haber menos de 5 ciclistas, lanzar una excepción de usuario sin nombre que será recogida en el bloque anónimo.

CREATE OR REPLACE FUNCTION f_cantidadCiclistasEquipo (V_EQUIPO IN CICLISTA.NOMEQ%TYPE)
RETURN NUMBER
IS
  V_NUM_CICLISTAS NUMBER;
BEGIN
  SELECT COUNT(*) INTO V_NUM_CICLISTAS
  FROM CICLISTA
  WHERE NOMEQ = V_EQUIPO;

  IF V_NUM_CICLISTAS < 5 THEN
    RAISE_APPLICATION_ERROR (-20006, 'hay menos de 5 ciclistas en este equipo, son muy pocos...');
  END IF;

  RETURN V_NUM_CICLISTAS;
END;
```

```
Ej3_CICLISMO12 x
11
```

```
-- bloque anónimo
DECLARE
  V_NOM_EQUIPO CICLISTA.NOMEQ%TYPE;
  V_NUM_CICLISTAS NUMBER;

  EQUIPO_PEQUEÑO EXCEPTION;
  PRAGMA EXCEPTION_INIT(EQUIPO_PEQUEÑO, -20006);

BEGIN
  V_NOM_EQUIPO := '&EQUIPO';

  V_NUM_CICLISTAS := f_cantidadCiclistasEquipo(V_NOM_EQUIPO);

  DBMS_OUTPUT.PUT_LINE(V_NUM_CICLISTAS);

EXCEPTION
  WHEN EQUIPO_PEQUEÑO THEN
    DBMS_OUTPUT.PUT_LINE('hay menos de 5 ciclistas en este equipo, son muy pocos...');

END;
```

```
Salida de DBMS Ejercicios_de_Refuerzo.sql CICLISTA
Tamaño de Buffer: 20000
Ej3_CICLISMO12 x
hay menos de 5 ciclistas en este equipo, son muy pocos...
```

Ejemplo con “Amore Vita”



## 9. Crea una función con un cursor implícito.

-- inspirado en el apartado 8

-- -- 8.1. Crea una función que acepte un nombre de equipo y devuelva la cantidad de integrantes del equipo,  
-- que tengan una edad superior a la media.

```
CREATE OR REPLACE FUNCTION f_cantidadCiclistasEquipo_v2 (V_NOM_EQUIPO IN
CICLISTA.NOMEQ%TYPE)
RETURN NUMBER
```

```
IS
  V_NUM_CICLISTAS NUMBER;

BEGIN
  SELECT COUNT(*) INTO V_NUM_CICLISTAS
  FROM CICLISTA, PUERTO
  WHERE CICLISTA.NOMEQ = V_NOM_EQUIPO
  AND CICLISTA.DORSAL = PUERTO.DORSAL
  AND CICLISTA.EDAD > (
    SELECT AVG(CICLISTA.EDAD)
    FROM CICLISTA
  );

  RETURN V_NUM_CICLISTAS;

END;
```

-- bloque anónimo

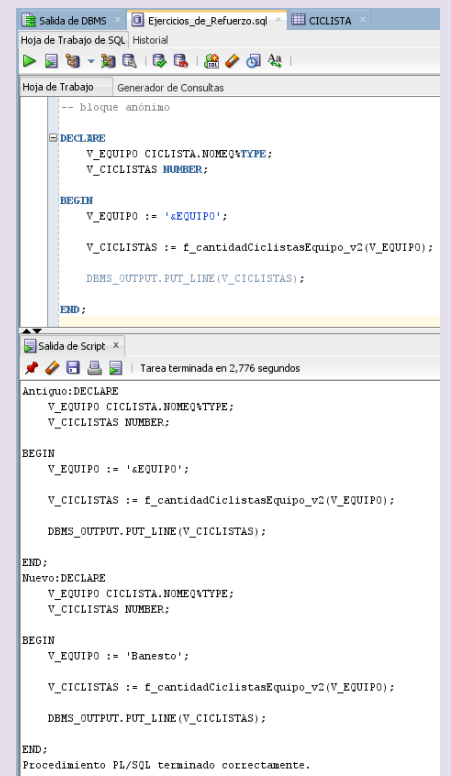
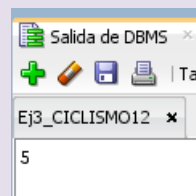
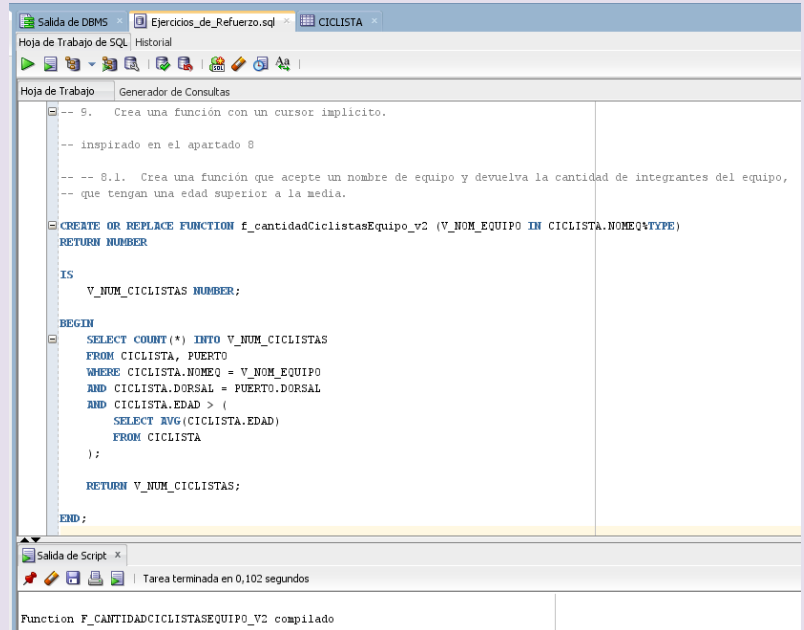
```
DECLARE
  V_EQUIPO CICLISTA.NOMEQ%TYPE;
  V_CICLISTAS NUMBER;

BEGIN
  V_EQUIPO := '&EQUIPO';

  V_CICLISTAS := f_cantidadCiclistasEquipo_v2(V_EQUIPO);

  DBMS_OUTPUT.PUT_LINE(V_CICLISTAS);

END;
```



## 10. Crea un procedimiento con un cursor explícito. Escríbelo con los dos tipos de cursor explícito posibles.

-- Replico el ejer 4 con bucle FOR

-- 4. Crea un procedimiento que acepte una categoría de puerto y liste todos los puertos de esa categoría.

-- Utiliza ambos tipos de cursor explícito.

```
CREATE OR REPLACE PROCEDURE p_listarPuertosCategoria_v2(V_CATEGORIA IN
PUERTO.CATEGORIA%TYPE)
```

IS

```
V_PUERTOS PUERTO%ROWTYPE;
```

```
CURSOR c_datos1 IS
```

```
SELECT * INTO V_PUERTOS
```

```
FROM PUERTO
```

```
WHERE CATEGORIA = V_CATEGORIA;
```

BEGIN

```
FOR V_PUERTOS IN c_datos1 LOOP
```

```
DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' ||
V_PUERTOS.NOMPUERTO);
```

```
DBMS_OUTPUT.PUT_LINE('Altura puerto: ' ||
V_PUERTOS.ALTURA);
```

```
DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' ||
V_PUERTOS.CATEGORIA);
```

```
DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' ||
V_PUERTOS.PENDIENTE);
```

```
DBMS_OUTPUT.PUT_LINE('Numero etapa: ' ||
V_PUERTOS.NETAPA);
```

```
DBMS_OUTPUT.PUT_LINE('Dorsal' ||
V_PUERTOS.DORSAL);
```

```
END LOOP;
```

```
END p_listarPuertosCategoria_v2;
```

-- bloque anónimo

DECLARE

```
V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;
```

BEGIN

```
V_CATEGORIA1 := '&CATEGORIA';
```

```
p_listarPuertosCategoria_v2(V_CATEGORIA1);
```

```
END p_listarPuertosCategoria_v2;
```

```
-- 10. Crea un procedimiento con un cursor explícito. Escríbelo con los dos tipos de cursor explícito posibles.
-- Replico el ejer 4 con bucle FOR
-- 4. Crea un procedimiento que acepte una categoría de puerto y liste todos los puertos de esa categoría.
-- Utiliza ambos tipos de cursor explícito.

CREATE OR REPLACE PROCEDURE p_listarPuertosCategoria_v2(V_CATEGORIA IN PUERTO.CATEGORIA%TYPE)
IS
    V_PUERTOS PUERTO%ROWTYPE;

    CURSOR c_datos1 IS
        SELECT * INTO V_PUERTOS
        FROM PUERTO
        WHERE CATEGORIA = V_CATEGORIA;

BEGIN
    FOR V_PUERTOS IN c_datos1 LOOP
        DBMS_OUTPUT.PUT_LINE('Nombre puerto: ' || V_PUERTOS.NOMPUERTO);
        DBMS_OUTPUT.PUT_LINE('Altura puerto: ' || V_PUERTOS.ALTURA);
        DBMS_OUTPUT.PUT_LINE('Categoria puerto: ' || V_PUERTOS.CATEGORIA);
        DBMS_OUTPUT.PUT_LINE('Pendiente puerto: ' || V_PUERTOS.PENDIENTE);
        DBMS_OUTPUT.PUT_LINE('Numero etapa: ' || V_PUERTOS.NETAPA);
        DBMS_OUTPUT.PUT_LINE('Dorsal' || V_PUERTOS.DORSAL);
    END LOOP;

END p_listarPuertosCategoria_v2;
```

Salida de DBMS: Ejercicios\_de\_Refuerzo.sql

Hoja de Trabajo: Generador de Consultas

Salida de Script: Tarea terminada en 0,092 segundos

Procedure P\_LISTARPUERTOSCATEGORIA\_V2 compilado

```
-- bloque anónimo

DECLARE
    V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;

BEGIN
    V_CATEGORIA1 := '&CATEGORIA';

    p_listarPuertosCategoria_v2(V_CATEGORIA1);

END p_listarPuertosCategoria_v2;

Nuevo:DECLARE
    V_CATEGORIA1 PUERTO.CATEGORIA%TYPE;

BEGIN
    V_CATEGORIA1 := '1';

    p_listarPuertosCategoria_v2(V_CATEGORIA1);

END p_listarPuertosCategoria_v2;

Procedimiento PL/SQL terminado correctamente.
```

Salida de Script: Tarea terminada en 3,371 segundos

Salida de DBMS: Ejercicios\_de\_Refuerzo.sql

Ej3\_CICLISMO12

```
Nombre puerto: Alto del Naranco
Altura puerto: 565
Categoria puerto: 1
Pendiente puerto: 6,9
Numero etapa: 10
Dorsal130
Nombre puerto: Coll de la Comella
Altura puerto: 1362
Categoria puerto: 1
Pendiente puerto: 8,07
Numero etapa: 10
Dorsal12
Nombre puerto: Navacerrada
Altura puerto: 1860
Categoria puerto: 1
Pendiente puerto: 7,5
Numero etapa: 19
Dorsal12
Nombre puerto: Puerto de Alisas
Altura puerto: 672
Categoria puerto: 1
Pendiente puerto: 5,8
Numero etapa: 15
Dorsal11
Nombre puerto: Puerto de Mijares
Altura puerto: 1525
Categoria puerto: 1
Pendiente puerto: 4,9
Numero etapa: 18
Dorsal124
Nombre puerto: Puerto de Pedro Bernardo
Altura puerto: 1250
Categoria puerto: 1
Pendiente puerto: 4,2
Numero etapa: 18
Dorsal125
```