

PAC DESARROLLO

CFGS Desarrollo de Aplicaciones Web

Módulo 2B: Bases de Datos

UF3. Lenguajes SQL: DCL y extensión procedimental

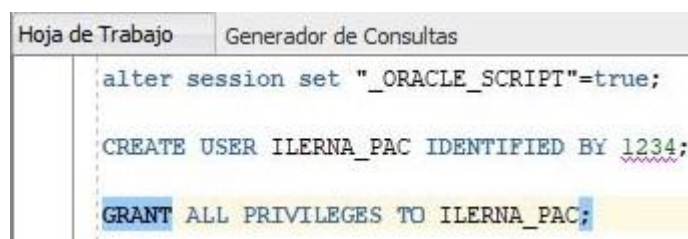


**David Nuñez
Merino**

2S2022 – DAW

CONFIGURACIÓN INICIAL

Antes de continuar con las actividades creamos una nueva conexión llamada “PAC_UF3” usando el usuario administrador y después creo un nuevo usuario “ILERNA_PAC” y contraseña “1234” y le asigno todos los privilegios:

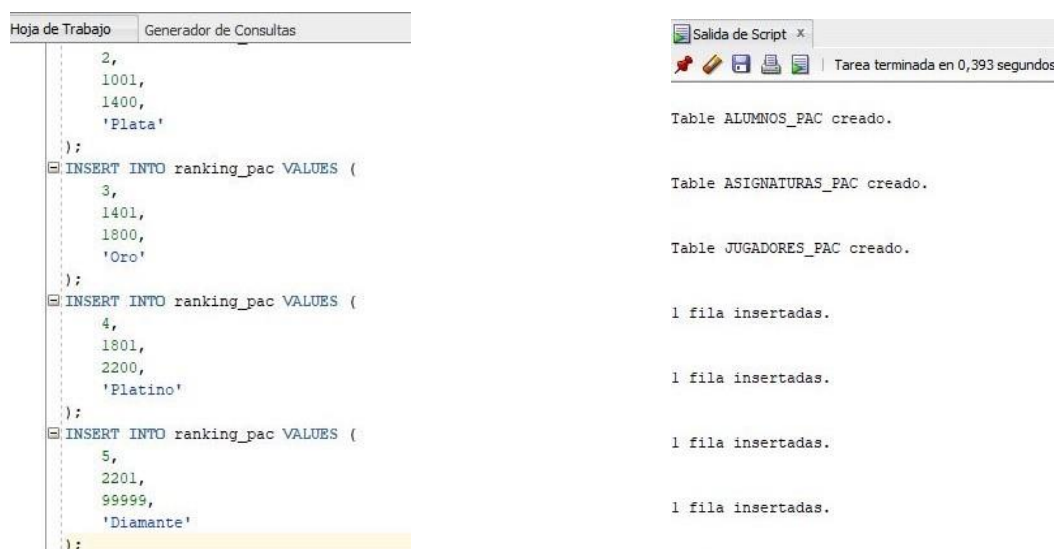


```
alter session set "_ORACLE_SCRIPT"=true;

CREATE USER ILERNA_PAC IDENTIFIED BY 1234;

GRANT ALL PRIVILEGES TO ILERNA_PAC;
```

A continuación, ejecutamos el script “PAC_Desarrollo_UF3_Configuracion Inicial.sql” para tener las tablas y registros que necesitamos para las practicas.



```
2,
1001,
1400,
'Plata'
);
INSERT INTO ranking_pac VALUES (
3,
1401,
1800,
'Oro'
);
INSERT INTO ranking_pac VALUES (
4,
1801,
2200,
'Platino'
);
INSERT INTO ranking_pac VALUES (
5,
2201,
99999,
'Diamante'
);
```

Salida de Script x

Tarea terminada en 0,393 segundos

Table ALUMNOS_PAC creado.

Table ASIGNATURAS_PAC creado.

Table JUGADORES_PAC creado.

1 fila insertadas.

1 fila insertadas.

1 fila insertadas.

1 fila insertadas.

EJERCICIOS

1) GESTIÓN DE USUARIOS Y TABLAS

1. Crear un usuario llamado "GESTOR" y contraseña "1234":

- Asígale los privilegios necesarios para que aparte de conectare. le permitan modificar, añadir o eliminar campos (No los registros) de las tablas Alumnos y Asignaturas.
- Conéctate con el usuario "GESTOR" y realiza lo siguiente con este usuario:
 - o En la tabla alumnos añade un nuevo campo "CIUDAD" - VARCHAR(30)
 - o En la tabla asignaturas modifica el campo "NOMBRE_PROFESOR" a VARCHAR(50)
 - o En la tabla asignaturas elimina el campo "CREDITOS"
 - o En la tabla asignaturas añade un nuevo campo llamado "CICLO" - VARCHAR (3)

```
-- 1)  GESTIÓN DE USUARIOS Y TABLAS -----  
  
-- 1. Usuario "GESTOR"  
-- creamos usuarios, definimos privilegios y modificamos tablas  
CREATE USER GESTOR IDENTIFIED BY 1234;  
GRANT CREATE SESSION TO GESTOR;  
GRANT ALTER, INSERT, DELETE ON ALUMNOS_PAC TO GESTOR;  
GRANT ALTER, INSERT, DELETE ON ASIGNATURAS_PAC TO GESTOR;  
  
ALTER TABLE ILERNA_PAC.ALUMNOS_PAC ADD CIUDAD VARCHAR(30);  
ALTER TABLE ILERNA_PAC.ASIGNATURAS_PAC MODIFY nombre_profesor VARCHAR(50);  
ALTER TABLE ILERNA_PAC.ASIGNATURAS_PAC DROP column credits;  
ALTER TABLE ILERNA_PAC.ASIGNATURAS_PAC ADD CICLO VARCHAR(3);
```

2. Crea un rol llamado "ROL_DIRECTOR" y un nuevo usuario llamado "DIRECTOR" y contraseña "1234"

- Al nuevo rol añádele los privilegios necesarios para que aparte de conectare, permitan seleccionar, insertar y modificar registros de las tablas alumnos y asignaturas.
- Asigna el nuevo rol "ROL_DIRECTOR" al usuario "DIRECTOR"
- Conéctate con el usuario "DIRECTOR" y realiza lo siguiente:
 - o Insertar un registro en la tabla Alumnos con tus datos
 - Id_alumno = (Las 2 primeras letras de tu nombre y tus apellidos junta)
 - Ejemplo: Juan Soria Morales JUSOMO
 - o Insertar un registro en la tabla Asignaturas con los datos de esta asignatura
 - Id_asignatura = 'DAX_M02B' ▪ Nombre_asignatura = 'MP2. Bases de datos B'
 - Nombre_profesor = (Nombre y apellidos del profesor actual)
 - Ciclo = 'DAX'

o Modificar el Ciclo del registro de la tabla Asignaturas insertado anteriormente

- Ciclo = (Poner DAM o DAW según te corresponde)

```
-- 2. Usuario "DIRECTOR"

CREATE ROLE ROL_DIRECTOR;
CREATE USER DIRECTOR IDENTIFIED BY 1234;
GRANT CREATE SESSION TO DIRECTOR;

GRANT SELECT,INSERT,UPDATE,DELETE ON ALUMNOS_PAC TO ROL_DIRECTOR;
GRANT SELECT,INSERT,UPDATE,DELETE ON ASIGNATURAS_PAC TO ROL_DIRECTOR;
GRANT ALTER ON ALUMNOS_PAC TO ROL_DIRECTOR;
GRANT ALTER ON ASIGNATURAS_PAC TO ROL_DIRECTOR;
GRANT ROL_DIRECTOR TO DIRECTOR;

INSERT INTO ILERNA_PAC.ALUMNOS_PAC VALUES ('DANUME', 'DAVID', 'NUÑEZ MERINO', '42', 'BARCELONA');
INSERT INTO ILERNA_PAC.ASIGNATURAS_PAC VALUES ('DAX_M02B', 'MP2 Bases de datos B', 'Guillem Mauri', 'DAX');
UPDATE ILERNA_PAC.ASIGNATURAS_PAC SET CICLO='DAM';
```

2) BLOQUES ANONIMOS

1. Crea un bloque anónimo donde se declare una variable constante llamada “puntos_actuales” del tipo NUMBER(10, 2) inicializa la variable con un valor de puntos como el de los jugadores creados en la configuración inicial. Muestra en qué ranking estaría ese jugador y una vez hecho incremente esos puntos en 300, realiza esta acción 3 veces más.

```
-- 2) BLOQUES ANONIMOS -----

-- PUNTOS ACTUALES

set SERVEROUTPUT on size 1000000;

DECLARE -- declaramos variable para acumular puntos y variable para contador del bucle
    puntos_actuales NUMBER(10,2) := 1000;
    contador NUMBER := 0;

BEGIN
    WHILE contador <= 3 LOOP -- recorremos las condiciones 3 veces en bucle para imprimir los resultados
        IF puntos_actuales BETWEEN 1 AND 1000 THEN
            DBMS_OUTPUT.PUT_LINE('Puntos actuales: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE('Ranking: Bronce ');
            puntos_actuales := puntos_actuales + 300;
            DBMS_OUTPUT.PUT_LINE('Incremento de 300: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE(' ');
        ELSIF puntos_actuales BETWEEN 1001 AND 1400 THEN
            DBMS_OUTPUT.PUT_LINE('Puntos actuales: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE('Ranking: Plata ');
            puntos_actuales := puntos_actuales + 300;
            DBMS_OUTPUT.PUT_LINE('Incremento de 300: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE(' ');
        ELSIF puntos_actuales BETWEEN 1401 AND 1800 THEN
            DBMS_OUTPUT.PUT_LINE('Puntos actuales: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE('Ranking: Oro ');
            puntos_actuales := puntos_actuales + 300;
            DBMS_OUTPUT.PUT_LINE('Incremento de 300: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE(' ');
        ELSIF puntos_actuales BETWEEN 1801 AND 2200 THEN
            DBMS_OUTPUT.PUT_LINE('Puntos actuales: ' || puntos_actuales);
            DBMS_OUTPUT.PUT_LINE('Ranking: Platino ');
            puntos_actuales := puntos_actuales + 300;
            DBMS_OUTPUT.PUT_LINE('Incremento de 300: ' || puntos_actuales);
        END IF;
        contador := contador + 1;
    END LOOP;
END;
```

Para resolver el ejercicio, declaramos dos variables, un contador para contar las veces que se ejecuta el bucle y otra puntos actuales que iremos modificando en cada iteración.

El resultado es el siguiente:

```
Puntos actuales: 1000
Ranking: Bronce
Incremento de 300: 1300
```

```
Puntos actuales: 1300
Ranking: Plata
Incremento de 300: 1600
```

```
Puntos actuales: 1600
Ranking: Oro
Incremento de 300: 1900
```

```
Puntos actuales: 1900
Ranking: Platino
Incremento de 300: 2200
```

```
Procedimiento PL/SQL terminado correctamente.
```

3) PROCEDIMIENTOS Y FUNCIONES SIMPLES

1. Crea una función llamada “NUMERO_MAYOR” que devuelva el mayor de 3 números pasados como parámetros, en casos de que se repita algún número, se ha de gestionar una excepción de error, diciendo “No se pueden repetir números en la secuencia”.

```
-----
-- 3)  PROCEDIMIENTOS Y FUNCIONES SIMPLES -----
-----

-- NUMERO MAYOR

CREATE OR REPLACE FUNCTION NUMERO_MAYOR -- creamos funcion y definimos variables
(primerNumero number, segundoNumero number, tercerNumero number)
RETURN number
IS
mayor number; -- la variable resultado

BEGIN

IF primerNumero = segundoNumero -- comprobamos si hay numeros repetidos
THEN
    IF segundoNumero = tercerNumero
    THEN
        RAISE_APPLICATION_ERROR(-2000,'No se pueden repetir números en la secuencia');
    END IF;
END IF;

IF primerNumero >= segundoNumero -- comparamos 2 numeros y guadamos el mayor
THEN
    mayor := primerNumero;
ELSE
    mayor := segundoNumero;
END IF;

IF mayor >= tercerNumero -- comparamos el mayor con el numero que falta
THEN
    mayor := mayor;
ELSE
    mayor := tercerNumero;
END IF;

RETURN mayor; -- devolvemos resultado

END;
/
```


Para realizar el ejercicio definimos la función NUMERO_MAYOR con tres variables que son los tres números que queremos comparar y una variable llamada "mayor" que era la que guardará el número mayor.

En primer lugar, realizamos unas comparaciones para comprobar que no tenemos ningún número repetido y después comparamos los dos primeros números y guardamos el mayor, a continuación, comparamos el almacenado con el último número y determinamos así cual es el mayor de los tres.

4) PROCEDIMIENTOS Y FUNCIONES COMPLEJAS

1. Crea una función llamada "JUGADORES_POR_RANKING" qué, dado un nombre de ranking pasado por parámetro, devuelva el total de jugadores que se encuentran en ese mismo ranking.

```
-- 4)  PROCEDIMIENTOS Y FUNCIONES COMPLEJAS -----  
  
--  NUMERO DE JUGADORES POR RANKING  
  
CREATE OR REPLACE FUNCTION JUGADORES_POR_RANKING --creamos funcion y definimos variable  
(nombre_ranking VARCHAR2)  
RETURN number  
IS  
total_jugadores number; -- variable resultado  
  
BEGIN -- contamos las coincidencias  
SELECT COUNT(*) INTO total_jugadores FROM JUGADORES_PAC WHERE ranking = nombre_ranking;  
RETURN total_jugadores; -- devolvemos el numero de coincidencias  
END;  
/
```

En este ejercicio, creamos una función llamada JUGADORES_POR_RANKING con la variable nombre_ranking y total_jugadores que guardara el resultado.

Para ello buscamos las coincidencias del ranking que definimos en la variable nombre_ranking en la tabla JUGADORES_PAC y las guardamos en la variable total_jugadores.

5) GESTIÓN DE TRIGGERS

1. Crea un trigger llamado “CAMBIO_RANKING_JUGADOR” que, al modificar la puntuación de un jugador, si la modificación implica un cambio en el ranking, se cambie automáticamente el ranking del jugador, ejemplo, pasar de tener 1390 puntos a 1410 puntos, el ranking del jugador cambiará automáticamente del que tenía en ese momento, ‘Plata’, al siguiente disponible, ‘Oro’.

```
-- 5)  GESTIÓN DE TRIGGERS -----  
  
-- CAMBIO DE RANKING DEL JUGADOR  
  
CREATE OR REPLACE TRIGGER CAMBIO_RANKING_JUGADOR -- creamos el trigger  
BEFORE UPDATE OF PUNTOS ON JUGADORES_PAC -- se ejecutara despues de actualizar la tabla  
FOR EACH ROW WHEN (new.puntos != old.puntos) -- cada vez que varien los puntos  
  
BEGIN -- con condiciones cambiamos el valor de la tabla  
IF :new.puntos BETWEEN 0 AND 1000 THEN  
    :new.ranking := 'Bronce';  
ELSIF :new.puntos BETWEEN 1001 AND 1400 THEN  
    :new.ranking := 'Plata';  
ELSIF :new.puntos BETWEEN 1401 AND 1800 THEN  
    :new.ranking := 'Oro';  
ELSIF :new.puntos BETWEEN 1801 AND 2200 THEN  
    :new.ranking := 'Platino';  
ELSIF :new.puntos BETWEEN 2201 AND 9999 THEN  
    :new.ranking := 'Diamante';  
END IF;  
END;
```

Crearemos el trigger CAMBIO_RANKING_JUGADOR que se ejecutara cuando se modifique el valor PUNTOS de cualquier fila de la tabla JUGADORES_PAC. Esto ocurrirá cuando el valor nuevo de puntos sea diferente del valor antiguo (antes de la modificación).

Cuando esto ocurra, comprobaremos el rango en el que se encuentra el nuevo valor de puntos y según cuál de las condiciones se cumplan le asignaremos un nuevo ranking.

6) BLOQUES ANÓNIMOS PARA PRUEBAS

1. **COMPROBACIÓN GESTIÓN USUARIOS Y TABLAS** Crea un bloque anónimo que muestre el registro de la tabla “alumnos_pac” y el de la tabla “asignaturas_pac”.

```
-- 6)  BLOQUES ANÓNIMOS PARA PRUEBAS DE CÃDIGO -----  
  
-- 1.  COMPROBACIÓN REGISTROS DE TABLAS  
/  
EXECUTE dbms_output.put_line('-- 1. COMPROBACIÓN REGISTROS DE TABLAS');  
-- mostramos el contenido de las tablas  
SELECT * FROM ILERNA_PAC.ALUMNOS_PAC;  
SELECT * FROM ILERNA_PAC.ASIGNATURAS_PAC;
```

Para mostrar el registro ejecutaremos un SELECT de todos los registros de la tabla ALUMNOS_PAC Y ASIGNATURAS_PAC.

2. COMPROBACIÓN DE LA FUNCION “NUMERO_MAYOR”

Crea un bloque anónimo que use la función “NUMERO_MAYOR”.

- Números de la prueba: 23, 37, 32
- Salida por pantalla: “El mayor entre (23, 37, 32) es: 37”

```
-- 2.  COMPROBACIÃ“N DE LA FUNCION NUMERO_MAYOR?
/
EXECUTE dbms_output.put_line('-- 2. COMPROBACIÃ“N DE LA FUNCION NUMERO_MAYOR?');

DECLARE -- declaramos variables
primerNumero NUMBER(2);
segundoNumero NUMBER(2);
tercerNumero NUMBER(2);
mayor NUMBER(2);

BEGIN --les damos valor
primerNumero := 23;
segundoNumero := 37;
tercerNumero := 32;

mayor := NUMERO_MAYOR (primerNumero, segundoNumero, tercerNumero); -- ejecutamos la funcion y mostramos el resultado por pantalla
DBMS_OUTPUT.PUT_LINE('El mayor entre ('||primerNumero||','||segundoNumero||','||tercerNumero||') es: '||mayor||');
END;
/
```

Para resolver este ejercicio declaramos las tres variables de los números que usa la función NUMERO_MAYOR y un más para almacenar el resultado.

Se definen las variables con los valores 23, 37 y 32. La variable Mayor será el resultado de la ejecución de la función NUMERO_MAYOR con el valor de las tres variables.

En la salida de texto mostraremos los tres valores y el resultado alojado en la variable mayor que es 37.

```
-- 2.  COMPROBACIÃ“N DE LA FUNCION NUMERO_MAYOR?
```

```
Procedimiento PL/SQL terminado correctamente.
```

```
El mayor entre (23,37,32) es: 37
```

```
Procedimiento PL/SQL terminado correctamente.
```


3. COMPROBACIÓN DE LA FUNCION “JUGADORES_POR_RANKING”

Crea un bloque anónimo que use la función “JUGADORES_POR_RANKING”.

- Nombre del ranking: Plata
- Salida por pantalla: “En el ranking Plata, tenemos a 2 jugadores.”

```
-- 3.  COMPROBACIÓN DE LA FUNCION JUGADORES_POR_RANKING?
/
EXECUTE dbms_output.put_line('-- 3. COMPROBACIÓN DE LA FUNCION JUGADORES_POR_RANKING?');

DECLARE --declaramos variables
nombre_ranking VARCHAR(20);
total_jugadores NUMBER(2);

BEGIN
nombre_ranking := 'Plata'; --definimos el parametro a buscar y sacamos el resultado por pantalla
total_jugadores := JUGADORES_POR_RANKING (nombre_ranking);
DBMS_OUTPUT.PUT_LINE('En el ranking '||nombre_ranking||' tenemos a '||total_jugadores||' jugadores.');
```

En este ejercicio declaramos dos variables, nombre_ranking donde definimos el ranking a buscar y total_jugadores donde guardaremos el número de coincidencias.

Definimos nombre_ranking con el valor “Plata” y total_jugadores como el resultado de la ejecución de la función JUGADORES_POR_RANKING y la variable nombre_ranking.

Mostraremos por pantalla el texto y el nombre del ranking y el número de coincidencias que se han encontrado.

```
-- 3.  COMPROBACIÓN DE LA FUNCION JUGADORES_POR_RANKING?

Procedimiento PL/SQL terminado correctamente.
En el ranking Plata tenemos a 3 jugadores.

Procedimiento PL/SQL terminado correctamente.
```

4. COMPROBACIÓN DE LOS TRIGGERS “CAMBIO_RANKING_JUGADOR”

Crea un bloque anónimo que pida por pantalla un id de un jugador y una nueva puntuación. Ha de actualizar el ranking del jugador si existe y comprobaremos que el trigger ha funcionado.

- Salida por pantalla: “El ranking del jugador (Nombre jugador) se ha modificado el día (Última Fecha y hora de modificación), antes era (Ranking Anterior) y ahora es (Nuevo Ranking)”

```
-- 4. COMPROBACIÓN DE LOS TRIGGERS
/
EXECUTE dbms_output.put_line('-- 4. COMPROBACIÓN DE LOS TRIGGERS');

DECLARE --declaramos las variables
id_jug NUMBER(2);
nueva_puntuacion NUMBER(20);
antiguo_ranking VARCHAR(20); --necesitamos almacenar el antiguo ranking
nuevo_ranking VARCHAR(20); -- el nuevo ranking
nombre2 VARCHAR(20); -- y el nombre de usuario

BEGIN
id_jug := &id_jug; --entrada de valores a modificar por pantalla
nueva_puntuacion := &nueva_puntuacion;

SELECT NOMBRE INTO nombre2 -- guardamos el nombre del jugador que modificamos
FROM JUGADORES_PAC
WHERE ID_JUGADOR = id_jug;

SELECT RANKING INTO antiguo_ranking -- guardamos antiguo ranking
FROM JUGADORES_PAC
WHERE ID_JUGADOR = id_jug;

UPDATE JUGADORES_PAC -- actualizamos con la nueva puntuacion
SET PUNTOS = NUEVA_PUNTUACION
WHERE ID_JUGADOR = id_jug;

SELECT RANKING INTO nuevo_ranking -- cogemos el nuevo ranking despues de la
FROM JUGADORES_PAC -- ejecucion del trigger
WHERE ID_JUGADOR = id_jug;
-- mostramos resultado por pantalla
DBMS_OUTPUT.PUT_LINE('El ranking del jugador '||nombre2||' se ha modificado el día '||sysdate||', antes era '||antiguo_ranking||' y ahora es '||nuevo_ranking||');
END;
/
```

Para comprobar los triggers primero tenemos que definir cinco variables:

- Id_jug donde guardaremos la id introducida por pantalla.
- nueva_puntuacion donde guardaremos la puntuación introducida por pantalla.
- Antiguo_ranking donde almacenaremos el ranking antes de modificarlo con el trigger.
- Nuevo_ranking donde almacenaremos el ranking después de modificarlo con el trigger.
- Nombre2 donde guardaremos el nombre correspondiente al id introducido.

Primero pediremos el id del jugador y la nueva puntuación por pantalla.

A continuación, buscaremos el nombre que corresponde a esa id en la tabla JUGADORES_PAC y lo almacenaremos en la variable nombre2.

Haremos lo mismo con el ranking que corresponde a esa id en la variable antiguo_ranking.

A continuación, realizaremos un UPDATE de la tabla JUGADORES_PAC y cambiaremos el valor de PUNTOS por la variable NUEVA_PUNTUACION del id que hemos introducido por pantalla.

Después guardaremos el nuevo ranking modificado por el trigger que hemos definido en la variable nuevo_ranking.

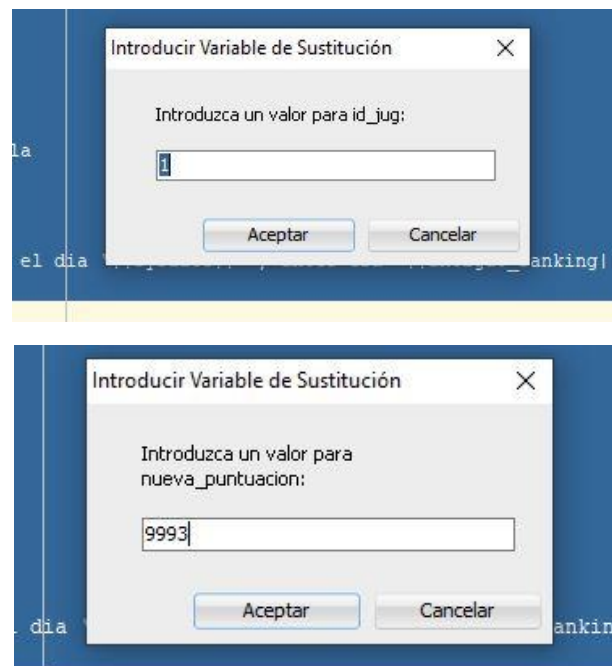
Por pantalla mostraremos el nombre que hemos guardado en la variable nombre2 la fecha y hora de modificación, el ranking guardado en antiguo_ranking y el ranking almacenado en nuevo_ranking.

Para que la fecha se muestre bien hemos introducido en la cabecera del script el siguiente código que nos dará la salida en día-mes-año y la hora con minutos y segundos que por defecto no nos muestra el sistema:

```
alter SESSION set NLS_DATE_FORMAT = 'DD-MM-YYYY HH24:MI:SS';
```

También en la cabecera del script introducimos el modificador set verify off para que no se nos muestren por la salida de script el código del bloque anónimo.

El resultado es este:



```
-- 4. COMPROBACIÓN DE LOS TRIGGERS
```

```
Procedimiento PL/SQL terminado correctamente.
```

```
El ranking del jugador Antonio se ha modificado el día 01-05-2022 17:10:38 , antes era Plata y ahora es Diamante
```

```
Procedimiento PL/SQL terminado correctamente.
```