Proyecto Grupal
Diseño y Gestión de Bases de Datos

Venta de tickets París

Curso 2023/2024

Ainhoa Rodríguez González
Alberto Ludeña Peirado
Joan Amorós Ramírez

TABLA DE CONTENIDO

1. DESCRIPCIÓN DEL SISTEMA DE INFORMACIÓN

Se desea gestionar la venta de tickets que realizan varias agencias en París para diferentes actividades turísticas y para el transporte a los diferentes aeropuertos de la ciudad; a continuación explicaremos cómo vamos a gestionar todos los procesos.

Los tickets pueden ser comprados en las agencias por uno o varios usuarios de los que se guarda su DNI, su nombre, primer apellido y la fecha de nacimiento. Cada agencia dispone de un CIF único, un nombre y una comisión asociada. Además, se registra la información de cada ticket, incluyendo el código del mismo, la fecha de compra y el precio, junto con el DNI del usuario y la agencia donde se ha adquirido.

Los productos disponibles para la venta poseen un código y un precio que varía según qué tipo de producto/servicio se compre (las actividades disponibles son las siguientes: bus a cada uno de los aeropuertos, bus turístico o crucero por el Sena). También se especifica la cantidad de cada producto en un ticket determinado. Es importante resaltar que todos los productos tienen como mínimo uno de los servicios o actividades asignadas.

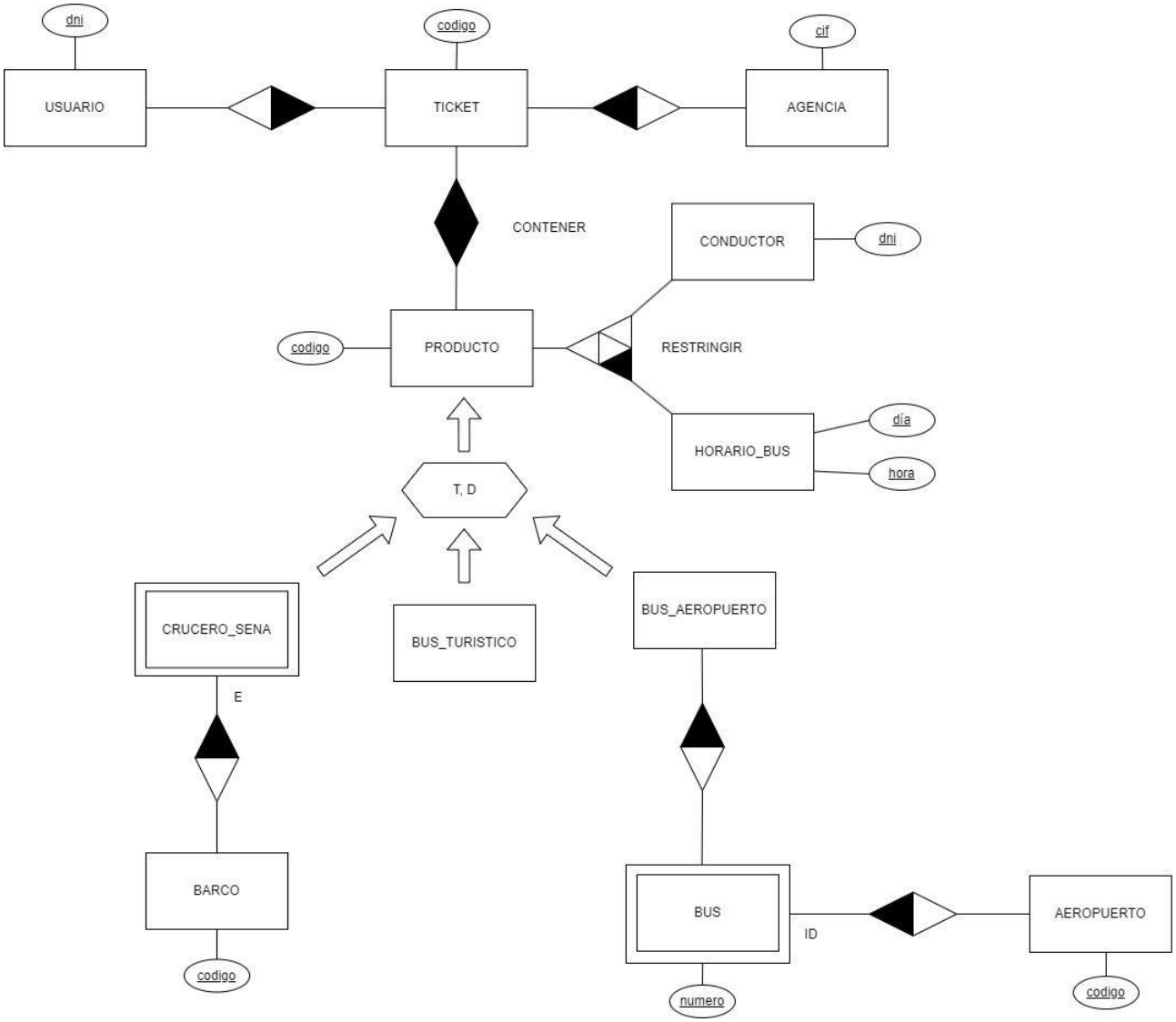
Por un lado, en los cruceros por el Sena se tiene registrado el tipo de actividad que se realiza en el crucero, la fecha y la hora a la que sale el crucero y el código del barco asociado. Cada barco tiene un código único y una capacidad máxima de pasajeros.

Por otra parte, en cuanto a los buses turísticos se guarda el número de plazas disponibles y el trayecto recorrido.

Con respecto a los aeropuertos y buses, cada aeropuerto tiene su código, nombre y distancia (en kilómetros) a la ciudad de París; cada bus está identificado por un número y está asociado a un aeropuerto específico (a través del código de aeropuerto) con detalles tales como el número de plazas, conductor y fecha de matriculación. Los buses a los aeropuertos hacen referencia a trayectos en autobús hacia un aeropuerto. Los productos son asociados (a través de su código) con un día de trayecto, un código de aeropuerto y un número de autobús.

Finalmente, se registra la información de los horarios de los buses, tanto el día como la hora, así como los detalles de cada conductor, que incluyen su DNI, nombre, apellidos y años de servicio. Además, se establecen restricciones para los conductores en relación con los productos y los horarios de los buses. De una manera más específica, se detalla qué conductores tienen limitaciones para ciertos productos en determinados días y horarios.

2. DIAGRAMA E-R (ENTIDAD-RELACIÓN)



3. ESQUEMA LÓGICO

USUARIO (dni, nombre, apellido, fechanacimiento)
C.P.: dni

AGENCIA (cif, nombre, comision)
C.P.: cif

TICKET (codigo, fechacompra, precio, dniusuario, cifagencia)
C.P.: codigo
C.Ajena: dniusuario → USUARIO
C.Ajena: cifagencia → AGENCIA

PRODUCTO (codigo, precio)
C.P.: codigo

CONTENER (codticket, codproducto, cantidad)
C.P.: codticket, codproducto
C.Ajena: codticket → TICKET
C.Ajena: codproducto → PRODUCTO

AEROPUERTO (codigo, nombre, distancia)
C.P.: codigo

BUS (numero, codaeropuerto, numplazas, nombreconductor, fechamatriculacion)
C.P.: numero, codaeropuerto
C.Ajena: codaeropuerto → AEROPUERTO

BUS_AEROPUERTO (codproducto, diatrayecto, codaeropuerto, numbus)
C.P.: codproducto
C.Ajena: codproducto → PRODUCTO
C.Ajena: codaeropuerto, numbus → BUS

BARCO (codigo, maxpasajeros)
C.P.: codigo

CRUCERO_SENA (codproducto, tipocrucero, fecha, hora, codbarco)
C.P.: codproducto
C.Ajena: codproducto → PRODUCTO
C.Ajena: codbarco → BARCO VNN [1]

BUS_TURISTICO (codproducto, trayecto, plazas)
C.P.: codproducto
C.Ajena: codproducto → PRODUCTO

HORARIO_BUS (dia, hora)
C.P.: dia, hora

CONDUCTOR (dni, nombre, apellido, anyosservicio)
C.P.: dni

RESTRINGIR (dniconductor, diahorario, horahorario, codproducto)
C.P.: dniconductor, diahorario, horahorario
C.Alternativa: codproducto, diahorario, horahorario
C.Ajena: diahorario, horahorario → HORARIO_BUS
C.Ajena: dniconductor → CONDUCTOR
C.Ajena: codproducto → PRODUCTO

[1] *VNN: Valor no nulo.*

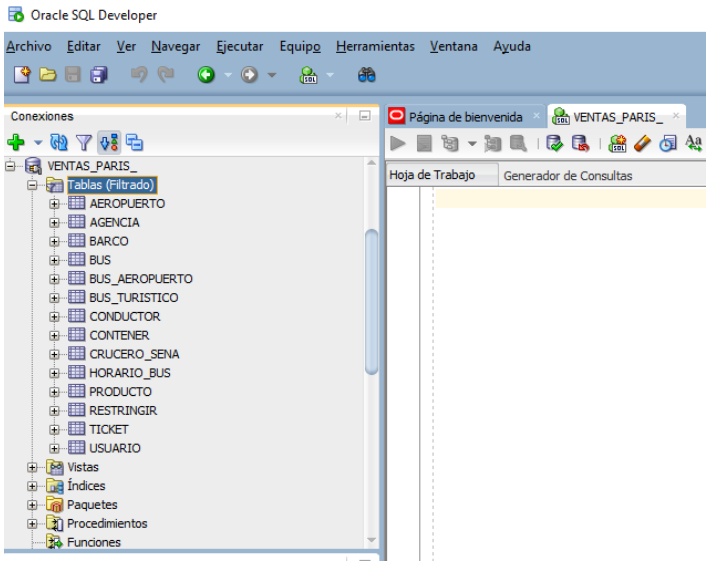
4. CAMBIOS A LA ESPECIFICACIÓN INICIAL DEL SISTEMA DE INFORMACIÓN

Para empezar, los cambios que se han realizado respecto a la versión inicial han sido dos.

El primer cambio realizado ha sido en el documento *crearbd.sql*, esto es debido a que, para poder crear las tablas e insertar los datos, era necesario crear un nuevo usuario. Así pues, hemos creado un nuevo usuario por medio de la línea de comandos SQL con nombre "*P1*" y con contraseña "*VENTA_PARIS*". Además, a parte de crearlo, también hemos tenido que otorgarle permisos de conexión y de recursos, tal como se muestra en la captura adjunta.

```
Copyright (c) 1982, 2014, Oracle.  All rights reserved.

SQL> CONNECT system/bdadmin
Connected.
SQL> CREATE USER VENTAS_PARIS IDENTIFIED BY PROYECTO;

User created.

SQL> GRANT CONNECT, RESOURCE TO VENTAS_PARIS;

Grant succeeded.

SQL> CONNECT VENTAS_PARIS/PROYECTO;
Connected.
```

Con este usuario, se han ejecutado en el programa SQL Developer cada una de las funciones, procedimientos y disparadores.



El segundo y último cambio que se ha realizado ha sido en el archivo *datos.sql*. En este archivo hemos añadido un usuario que es menor de edad porque era necesario para poder comprobar que el disparador "**VERIFICAR_EDAD_USUARIO**" realizase bien las acciones que se había implementado en este. Es decir, necesitábamos tener un usuario que sea menor de 18 años para que intente comprar un ticket y, así, poder comprobar que el disparador funciona correctamente y no deja que los menores de edad compren tickets..

El usuario que hemos añadido es:

INSERT INTO USUARIO values ('99999999Z', 'Usuario menor', 'Test', TO_DATE('01/01/2008', 'DD/MM/YYYY'));

Después, lo que hemos hecho ha sido agregar un ticket con el DNI '99999999Z' para la comprobación del disparador.

5. FUNCIONES, PROCEDIMIENTOS Y DISPARADORES

5.1. Funciones

En este apartado se muestran las funciones que han sido creadas, en este caso, se cuentan con 3 funciones que serán explicadas a continuación:

La primera función creada es "**CONTAR_BUSES**". Esta función toma un parámetro de entrada 'p_codaeropuerto', que es indicado por el usuario. Luego, se declaran cuatro variables: "v_count" para almacenar el número de buses, "v_total_plazas" para poder almacenar el número total de plazas en todos los buses, "v_conductor" que es para almacenar el nombre del conductor con el bus más antiguo y "v_ano_antiguo" que almacena el año de matriculación del bus más antiguo.

Después, se emplea la tabla BUS para contar la cantidad de buses asociados al aeropuerto especificado y se almacena el resultado en la variable "v_count".

Una vez hecho esto, se calcula la suma total de las plazas de todos los buses asociados y se vuelve a almacenar, esta vez en la variable "v_total_plazas". Finalmente, para encontrar el conductor, el bus más antiguo y la fecha asociada a estos dos, se realiza una consulta principal, donde se hará uso de la función EXTRACT, la cual permite extraer únicamente el año de la fecha de matriculación; y, asimismo, se realizará una subconsulta para obtener el nombre del conductor que conduce el autobús y la fecha de matriculación de este.

Finalmente, se devuelven los resultados obtenidos con la sentencia RETURN y, además, se lanza una excepción por si no se encuentra el código del aeropuerto.

La manera en la que ejecutaremos esta función será:

```
SELECT contar_buses('BVS') FROM DUAL;
SELECT contar_buses('ORL') FROM DUAL;
SELECT contar_buses('CDG') FROM DUAL;
```

Finalmente obtendremos como resultado lo siguiente:

```
SQL> SELECT contar_buses('BVS') FROM DUAL;

CONTAR_BUSES('BVS')
--------------------------------------------------------------------------------
Numero de buses: 1, Total de plazas: 33, Conductor con bus mas antiguo: Ruben, c
on fecha de matriculacion en el anyo: 2000


SQL> SELECT contar_buses('ORL') FROM DUAL;

CONTAR_BUSES('ORL')
--------------------------------------------------------------------------------
Numero de buses: 3, Total de plazas: 79, Conductor con bus mas antiguo: Yolanda,
 con fecha de matriculacion en el anyo: 1970


SQL> SELECT contar_buses('CDG') FROM DUAL;

CONTAR_BUSES('CDG')
--------------------------------------------------------------------------------
Numero de buses: 3, Total de plazas: 86, Conductor con bus mas antiguo: Marta, c
on fecha de matriculacion en el anyo: 1980
```

La segunda función implementada es “**TOTAL_VENTAS_USUARIO**”. En dicha función se crean dos variables: “total_ventas”, para almacenar la suma del precio de todos los tickets que el usuario ha comprado, y “usuario_existe”, donde si el usuario no existe se almacena un 0. Luego, se realiza una consulta inicial donde se comprueba que el dni que se ha especificado en el parámetro 'p_dni’ existe. Una vez se comprueba que el usuario existe, la función realiza otra consulta en la que se suma el precio de todos los tickets que este haya comprado y el resultado se almacena en la variable creada ‘total_ventas'.

Después, se imprime el valor de dicha variable junto con el siguiente mensaje: “(‘El total de ventas para el usuario con DNI’ || p_dni || ‘es: ‘ || total_ventas|| ‘euros.’”

Por otra parte, en el caso en el que el dni aportado no exista, la función imprime por pantalla el siguiente ERROR: “-20001, 'El usuario no existe.'”.

```
SQL> SELECT total_ventas_usuario('22222222A') FROM dual;

TOTAL_VENTAS_USUARIO('22222222A')
--------------------------------------------------------------------------------
El total de ventas para el usuario con DNI 22222222A es: 0 euros

SQL> SELECT total_ventas_usuario('74384681J') FROM dual;

TOTAL_VENTAS_USUARIO('74384681J')
--------------------------------------------------------------------------------
El total de ventas para el usuario con DNI 74384681J es: 792 euros

SQL> SELECT total_ventas_usuario('11111222A') FROM dual;

TOTAL_VENTAS_USUARIO('11111222A')
--------------------------------------------------------------------------------
El total de ventas para el usuario con DNI 11111222A es: 244,66 euros

SQL> SELECT total_ventas_usuario('22233334B') FROM dual;

TOTAL_VENTAS_USUARIO('22233334B')
--------------------------------------------------------------------------------
El total de ventas para el usuario con DNI 22233334B es: 538 euros
```

Por último, la última función creada es “**DETALLE_USUARIO**”. Lo que hace la función es, en primer lugar, guarda el dni del usuario del que se quiere consultar la información en el parámetro “p_dni”. A continuación, dentro de la función, se crean 3 variables: “detalles_tickets”, aquí se almacenará la información de los tickets, “total_tickets”, para guardar el número total de tickets que el usuario ha comprado y “total_gastado”, donde se almacenará la cifra total que el usuario ha gastado en todos los tickets; además de implementar un *cursor* que obtiene el código y el precio de los tickets que ha comprado el usuario especificado.

Seguidamente, se inicializa la variable “detalles_tickets”, es decir, la variable creada se iguala a una cadena de texto vacía porque, al tratarse de una variable de tipo VARCHAR2, para que esta no contenga “información basura” se realiza este proceso. Y se abre el cursor creado.

Después, se crea un bucle a través del cual se obtendrán los códigos y precios de los tickets que el usuario ha comprado, haciendo uso de la sentencia FETCH, cuya función es obtener la fila que se encuentra próxima al cursor y almacenarla, en este caso, en la variable “v_ticket”. En el caso en el que el código de los tickets no se encuentre, se saldrá y finalizará el bucle.

Entonces, se cierra el cursor, se utiliza la sentencia RTRIM() para eliminar la última coma y espacio del mensaje que se almacena en la variable “detalles_tickets” y que se mostrará por pantalla; y se comprueba si la variable ya nombrada es nula. Si se diese el caso, se imprimiria por pantalla al usuario que ha surgido un *error*, cuyo mensaje es el que se muestra a continuación: “-20003, 'El usuario no ha comprado ningún ticket.'.

Luego, se llevan a cabo dos consultas. En la primera se contabilizan el total de tickets que el usuario ha comprado y se guarda dicha cantidad en la variable “total_tickets”. Y en la segunda se suma el precio de todos los tickets y, así, calcular la cantidad total que este se ha gastado comprando los tickets, dicha información se almacena en la variable “total_gastado”. Finalmente, se devuelven las variables donde se han almacenado

los datos y se le muestra al usuario por pantalla el siguiente mensaje: “'El usuario con DNI ' || p_dni || ' ha comprado un total de ' || total_tickets || ' tickets por un total de ' || total_gastado || ' euros. Los tickets comprados son: ' || detalles_tickets”.

Por otra parte, también se ha definido en la función una excepción para que, si no se ha encontrado que el usuario haya realizado la compra de ningún ticket, se le muestre al usuario por pantalla dicho cometido, es decir, la siguiente advertencia: “El usuario no ha comprado tickets”.

En el siguiente ejemplo nos muestra lo que realiza la función:

```
DETALLE_USUARIO('74384681J')
--------------------------------------------------------------------------------
El usuario con DNI 74384681J ha comprado un total de 6 tickets por un total de 7
92 euros. Los tickets comprados son: AAA (123 euros), AAB (123 euros), AAC (123
euros), AAD (100 euros), CDZ (200 euros), ZZZ (123 euros)
```

En el siguiente ejemplo veremos la advertencia de el usuario en el caso de que no haya comprado ningún ticket:

```
ERROR at line 1:
ORA-20003: El usuario no ha comprado ning·n ticket.
ORA-06512: at "PROYECTO.DETALLE_USUARIO", line 29
```

5.2. Procedimientos

A continuación, hemos creado 4 procedimientos, los cuales se explican seguidamente:

El primer procedimiento se llama “**INSERTAR_USUARIO**”. Este procedimiento consiste en que el usuario introduce el dni, el nombre, el apellido y la fecha de nacimiento de un cliente concreto; esta información se guarda en los parámetros “dni”, “nombre”, “apellido” y “fechanacimiento”. Seguidamente, se crean 3 variables: “v_dni” y “v_fecha”, en estas se almacenará el dni del usuario que se quiere introducir en la base de datos y la fecha de nacimiento en formato fecha.

Después de esto, lo que hace el procedimiento es convertir los datos que se ha introducido en el parámetro “fechanacimiento” en un tipo de dato de fecha debido a que el usuario introduce la fecha de nacimiento en formato VARCHAR2 y la variable “fechanacimiento” de la tabla USUARIO está declarada como tipo fecha. Luego, verifica si el dni que se ha proporcionado se encuentra en la base de datos, si se da el caso en el que el usuario sí que existe se mostrará por pantalla un mensaje de ERROR, el cual es “-20001, 'El usuario con DNI ' || dni || ' ya existe.'”. Si, al contrario, el usuario no existe, se realizará una consulta en la que el procedimiento insertará los datos ya especificados en los parámetros y variables creadas.

Por otro lado, en el procedimiento también puede observarse que hay una *excepción*, la cual consiste en que, si no se encuentra el usuario, se sigan introduciendo los datos del mismo para crearlo.

Todo lo que se ha explicado anteriormente puede verse aplicado en la siguiente captura de pantalla, donde se inserta a un nuevo usuario de ejemplo:

```
SQL> EXECUTE INSERTAR_USUARIO('12345678A', 'Alberto', 'Ludena', '14/09/1999');

PL/SQL procedure successfully completed.

SQL> SELECT * FROM USUARIO;

DNI       NOMBRE                    APELLIDO                  FECHANAC
--------- ------------------------- ------------------------- --------
22222222A Juan                      Pérez                     01/01/06
12345678A Alberto                   Ludena                    14/09/99
```

El segundo procedimiento que se ha creado es “**CALCULAR_TOTAL_TICKETS**”, del que se puede destacar el uso de un *cursor*. El cursor definido en este selecciona el precio de todos los tickets de la tabla TICKET. Justo después de este, se declaran las variables “v_precio” y “v_total”, que se inicializa a 0; las cuales se utilizarán para guardar el precio de cada uno de los tickets y guardar el precio total de todos los tickets que haya.

A continuación, en el bloque principal del código, se abre el cursor y se implementa un bucle donde se va cogiendo el precio de cada tipo de ticket y se suma al valor que haya en la variable “v_total”, así hasta obtener la suma del precio de todos los tickets. Cuando ya no haya más tickets, el procedimiento saldrá del bucle y cerrará el cursor.

En el caso en el que la variable “v_total” sea igual a 0, se mostrará por pantalla un mensaje que dirá “ 'No hay tickets o el total del precio de todos los tickets es 0.' “, en otro caso se muestra el siguiente mensaje: “ 'El total del precio de todos los tickets es: ' || v_total ”.

En la imagen adjunta podemos ver que el precio total de todos los tickets es de 1574,66 euros. Si hacemos la consulta de la tabla ticket y sumamos el precio de todos los tickets, nos devolverá la cantidad que nos imprime nuestro procedimiento. Esto servirá para saber el beneficio que tenemos en cuanto a la venta de tickets.

```
SQL> EXECUTE calcular_total_tickets;
El total del precio de todos los tickets es: 1574,66

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TICKET;

CODIGO     FECHACOM     PRECIO DNIUSUARIO CIFAGENCIA
---------- -------- ---------- ---------- ----------
AAA        01/02/18        123 74384681J
AAB        02/02/18        123 74384681J
AAC        03/02/18        123 74384681J
AAD        02/02/18        100 74384681J  9999999
ABA        03/02/18      44,66 11111222A
ACA        05/10/18        200 22233334B  11111111
ADA        02/10/18        138 22233334B
CDA        07/01/19        200 22233334B  9999999
CXA        05/01/19        200 11111222A  11111111
CDZ        04/01/19        200 74384681J  9999999
ZZZ        01/02/18        123 74384681J
```

Posteriormente, otro procedimiento que se ha creado es para contar el número de tickets, el número de conductores y que realice el promedio de tickets que le tocan a cada uno de los conductores. Este procedimiento no toma ningún valor por parámetro y no devuelve ningún valor, el propósito de este procedimiento es realizar el cálculo y mostrar el resultado en la salida.

Lo que se está haciendo es ver cuantos tickets se asignan a cada conductor, para ello lo primero que se hace es calcular el total de tickets en la tabla TICKET y almacenar el resultado en la variable “v_total_tickets”. Después, se calcula el número de conductores en la tabla CONDUCTOR y se almacena el resultado en la variable “v_total_conductores”, si el total de conductores es mayor que 0, calcula el promedio de tickets por conductor dividendo el total de tickets entre el total de conductores. El resultado de esto se almacena en la variable “v_promedio”.

Después de realizar dicha acción, se imprime el total de tickets y el total de conductores, incluso si estos son 0. Si el promedio de tickets por conductor es mayor que 0, el procedimiento imprime el promedio.

Si se produce algún error durante la ejecución del procedimiento, se captura la excepción y se imprime un mensaje de error con la función SQLERRM, que devuelve el mensaje de error que esté asociado.

```
SQL> EXECUTE ticket_conductores;
Total de tickets: 11
Total de conductores: 3
Promedio de tickets por conductor: 3,67
```

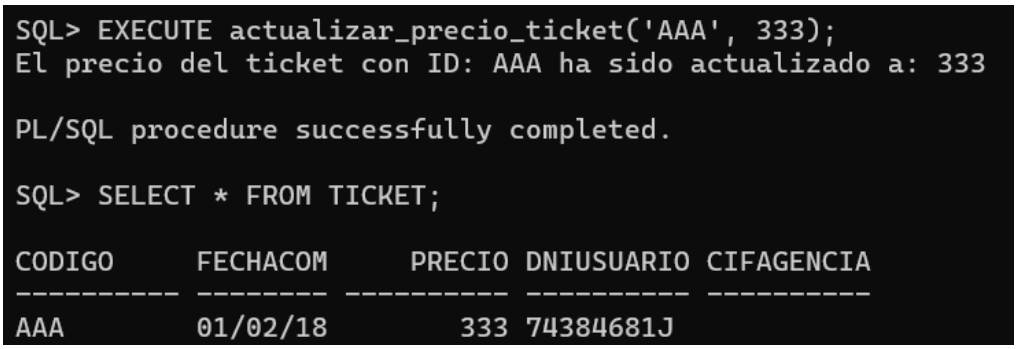
El último procedimiento que se ha creado es “**ACTUALIZAR_PRECIO_TICKET**” y lo que hace es que, el usuario, le pasa al procedimiento el código del ticket y el nuevo precio de este y este actualiza dicha información de la tabla TICKET para que cuando los usuarios vayan a comprar tickets, el precio de estos esté cambiado y no haya confusiones.

Por otra parte, el funcionamiento interno del procedimiento es que, primero, la información que introduce el cliente se almacena en dos parámetros: “p_codigo” y “p_nuevo_precio”, en los cuales se guarda el código del ticket que se quiere cambiar el precio y el nuevo valor del precio de dicho ticket.

Después, puede mostrarse el bloque de código principal, es decir, todo el código que se encuentra a partir de la sentencia BEGIN. Lo primero que hace el procedimiento en este bloque es, mediante una consulta SQL, actualizar el precio del ticket del que se ha especificado el código. Luego, nos encontramos con una condición IF, donde si el código del ticket indicado es igual a 0, salta un mensaje por pantalla que dice “ 'No se encontró el ticket con ID: ' || p_codigo “. Si esto no se cumple, entonces se muestra por pantalla el siguiente mensaje: “ 'El precio del ticket con ID: ' || p_codigo || ' ha sido actualizado a: ' || p_nuevo_precio “.

Por último, puede observarse que hay una *excepción*, esta consiste en que si no ocurre ninguno de los casos que están definidos en el procedimiento, este muestra un mensaje de ERROR por pantalla: “ 'Se produjo un error: ' || SQLERRM ”. Esta sentencia se utiliza para obtener el mensaje de error correspondiente a la última excepción que se ha producido en el bloque SQL.

Para comprobar que el procedimiento funciona correctamente, se introduce la instrucción EXECUTE en la línea de comandos SQL junto con el nombre del procedimiento y el código del ticket y el nuevo precio. Cuando se ha ejecutado el procedimiento se hace una consulta SQL para obtener todos los datos de la tabla TICKET y, así, comprobar que el precio se ha actualizado tal como se muestra en la imagen adjunta.



5.3. Disparadores

Finalmente, se han creado dos disparadores en este proyecto.

El primer disparador que se ha implementado es "**CHECK_CONDICIONES_AEROPUERTO**". Este disparador comprueba que, cuando se introduzca un aeropuerto, este cumpla con las condiciones establecidas, como por ejemplo, que el nombre tenga entre 3 y 25 caracteres, que la distancia del aeropuerto sea mayor que 0, que el código de este tenga entre 3 y 10 caracteres y que el nombre del aeropuerto solo contenga letras y espacios. No obstante, puede verse que dicho disparador trabaja con la tabla AEROPUERTO.

En cuanto a la parte más funcional, al principio del disparador se declaran 2 variables: "v_nombre_len", que recogerá la longitud de caracteres del nombre del aeropuerto, y "v_codigo_len", que almacena cuántos caracteres tiene el código especificado para el aeropuerto correspondiente. Luego, en la parte principal del código, se obtiene la longitud del nombre y del código del aeropuerto con la función *LENGTH()* y se guarda en las variables nombradas anteriormente. Una vez realizada la acción, se implementan una serie de condiciones donde se comprobará si la longitud del nombre almacenado en "v_nombre_len" está entre 3 y 25, si no es así, se mostrará por pantalla el siguiente mensaje de ERROR: "-20001, 'El nombre debe tener entre 3 y 25 caracteres.'".

También se comprueba que la distancia se mayor que 0, porque sino saltaría por pantalla el siguiente mensaje: "-20002, 'La distancia debe ser mayor que 0.'". Que el código esté comprendido entre los 3 y 10 caracteres debido a que si no es el caso, se mostrará por pantalla un mensaje de ERROR: "-20003, 'El código del aeropuerto debe tener entre 3 y 10 caracteres.'". Y además, en la última condición se comprueba que el nombre esté compuesto únicamente por letras y, también, por espacios. Si esto no ocurre así, se muestra por pantalla el siguiente ERROR: "-20004, 'El nombre solo puede contener letras y espacios.'"

A continuación, se muestran capturas de pantalla de todos los ejemplos que se han realizado en cada uno de los casos. Para este disparador, los datos que se introducen de ejemplo no cumplen con las especificaciones que se deben de comprobar, esto es así para ver si el trigger cumple bien su funcionalidad y notifica dichos errores.

1º) El nombre está entre 3 y 25 caracteres:

En este primer caso, el usuario quiere insertar un nuevo aeropuerto con nombre 'AB'. Sin embargo, por pantalla se muestra un mensaje de error, esto se debe a que el nombre que ha insertado el usuario tiene menos de 3 caracteres.

```
SQL> INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('AB', 100, 'ABC');
INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('AB', 100, 'ABC')
            *
ERROR at line 1:
ORA-20001: El nombre debe tener entre 3 y 25 caracteres.
ORA-06512: at "PROYECTO.CHECK_CONDICIONES_AEROPUERTO", line 9
ORA-04088: error during execution of trigger
'PROYECTO.CHECK_CONDICIONES_AEROPUERTO'
```

2º) La distancia tiene que ser mayor que 0

Ahora nos encontramos con otro caso en el que el usuario introduce un nuevo aeropuerto, especificando el nombre, la distancia y el código. No obstante, puede observarse que aparece un error por pantalla debido a que la distancia especificada es 0, cosa que no es posible como el propio mensaje de error indica.

```
SQL> INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto Test', 0, 'TEST');
INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto Test', 0, 'TEST')
            *
ERROR at line 1:
ORA-20002: La distancia debe ser mayor que 0.
ORA-06512: at "PROYECTO.CHECK_CONDICIONES_AEROPUERTO", line 13
ORA-04088: error during execution of trigger
'PROYECTO.CHECK_CONDICIONES_AEROPUERTO'
```

3º) La longitud del código del aeropuerto está entre 3 y 10

Luego, el usuario intenta insertar de nuevo el mismo aeropuerto que no había podido crear, esta vez especificando una distancia mayor que 0. Sin embargo, vuelve a saltar un mensaje porque el código que ha especificado ahora tiene menos de 3 caracteres.

```
SQL> INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto Test', 100, 'AB');
INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto Test', 100, 'AB')
            *
ERROR at line 1:
ORA-20003: El c%digo del aeropuerto debe tener entre 3 y 10 caracteres.
ORA-06512: at "PROYECTO.CHECK_CONDICIONES_AEROPUERTO", line 17
ORA-04088: error during execution of trigger
'PROYECTO.CHECK_CONDICIONES_AEROPUERTO'
```

4º) El nombre tienen que ser letras solo y puede admitir espacios

El último caso que se muestra es que el usuario cambia el código y el nombre del aeropuerto, pero como bien se indica en el error que le aparece en la pantalla, el nombre no puede contener ni número ni ningún caracter especial, solo letras.

```
SQL> INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto123', 100, 'TEST');
INSERT INTO AEROPUERTO (nombre, distancia, codigo) VALUES ('Aeropuerto123', 100, 'TEST')
            *
ERROR at line 1:
ORA-20004: El nombre solo puede contener letras y espacios.
ORA-06512: at "PROYECTO.CHECK_CONDICIONES_AEROPUERTO", line 21
ORA-04088: error during execution of trigger
'PROYECTO.CHECK_CONDICIONES_AEROPUERTO'
```

Finalmente, se muestra un ejemplo de cómo se ejecuta el disparador correctamente:

```
SQL> INSERT INTO AEROPUERTO (codigo, nombre, distancia) VALUES ('LAX', 'Los Angeles International', 300);

1 row created.

SQL> SELECT * FROM AEROPUERTO;

CODIGO     NOMBRE                     DISTANCIA
---------- ------------------------ ----------
BVS        BEAUVAIS                        107
ORL        ORLY                             17
CDG        CHARLES DE GAULLE                31
LAX        Los Angeles International       300
```

Por otra parte, el segundo trigger que se ha creado es “**VERIFICAR_EDAD_USUARIO**”. Sirve para comprobar que un usuario tenga la edad permitida, es decir, que sea mayor de 18 años cuando compre un ticket. Además, este disparador trabaja con la tabla TICKET.

No obstante, en el trigger, primero se declaran las variables “v_fecha_nacimiento”, para almacenar la fecha de nacimiento del cliente, “v_edad”, aquí se almacenará la edad de la persona, y “ex_menor_de_edad”, esta es una excepción personalizada que se utiliza para manejar los casos en los que el usuario es menor de 18 años.

Después, en la parte del BEGIN, se hace una consulta, a partir del dni del usuario que se quiere consultar, para obtener la fecha de nacimiento y guardarla en su respectiva variable. Posteriormente, se calcula la edad del usuario, restando a la fecha actual del sistema la fecha de nacimiento obtenida anteriormente y dividiendo el resultado entre el valor 365.25 días. Se pone dicho valor para tener en cuenta los años bisiestos. El valor final se almacena en la variable “v_edad”, que ya se creó al principio.

Hecho esto, se comprueba si el valor introducido en esta variable es menor que 18. Si esta condición se cumpliese, aparecería instantáneamente por pantalla el siguiente mensaje de ERROR: “-20001, 'El usuario es menor de 18 años' ”.

Por último, en la siguiente captura de pantalla que se adjunta a continuación, se muestra cómo el usuario con dni ‘99999999Z’ quiere comprar el ticket con código ‘T1’ y cuyo precio es de 100 unidades monetarias.A pesar de ello, por pantalla aparece un mensaje de error porque el usuario es menor de edad.

```
SQL> INSERT INTO TICKET (codigo, dniusuario, precio) VALUES ('T1', '99999999Z', 100);
INSERT INTO TICKET (codigo, dniusuario, precio) VALUES ('T1', '99999999Z', 100)
            *
ERROR at line 1:
ORA-20001: El usuario es menor de 18 anyos
ORA-06512: at "PROYECTO.VERIFICAR_EDAD_USUARIO", line 20
ORA-04088: error during execution of trigger 'PROYECTO.VERIFICAR_EDAD_USUARIO'
```

Ahora se adjunta una captura donde se consulta el dni del usuario para poder comprobar que este es menor y que el disparador hace su función correctamente.

```
SQL> SELECT * FROM USUARIO WHERE dni = '99999999Z';

DNI        NOMBRE                    APELLIDO                  FECHANAC
---------- ------------------------- ------------------------- --------
99999999Z  Usuario menor             Test                      01/01/08
```

6. OTROS ELEMENTOS

En este apartado no hemos incluido nada debido a que no nos hemos visto en la necesidad de agregar más usuarios.

7. BIBLIOGRAFÍA

Libros, apuntes,…:

Título: Diseño de bases de datos relacionales
Autor/-es: Yolanda Marhuenda
Año: 2023

Título: Sesión de prácticas. PL/SQL: Funciones
Autor/-es: Yolanda Marhuenda
Año: 2023

Título: Programación sobre bases de datos: PL / SQL
Autor/-es: Yolanda Marhuenda
Año: 2023

Título: Sesión de prácticas. PL/SQL: Procedimientos y cursores
Autor/-es: Yolanda Marhuenda
Año: 2023

Título: Sesión de prácticas. PL/SQL: Manejo de excepciones
Autor/-es: Yolanda Marhuenda
Año: 2023

Título: Sesión de prácticas. PL/SQL: Disparadores
Autor/-es: Yolanda Marhuenda
Año: 2023

Direcciones web:

Título: Función ‘EXTRACT’
https://learn.microsoft.com/es-es/azure/databricks/sql/language-manual/functions/extract

Título: Función ‘RTRIM’
https://ora.u440.com/cadena/rtrim.html