Trabajo 2. PL/SQL
Reservas festival

1. Descripción de la práctica

Se presenta el caso de uso de una empresa de eventos que organiza un festival de música y teatro. Como se trata de un festival muy demandado, y para que más gente pueda disfrutar de los espectáculos se ha creado un sistema de abonos. De manera que para acudir a los conciertos los asistentes pueden adquirir un abono que les permite asistir a un máximo de 10 eventos. Los abonos son personales y sólo se permite adquirir un bono por persona. Una vez adquirido el abono, cada usuario puede realizar la reserva de plaza en los 10 eventos que prefiera, haciendo la reserva de uno en uno. Para este caso de uso trabajaremos con las siguientes tablas:

1. ***Clientes*** *(NIF (PK), nombre, apellido1, apellido2)*
2. ***Abonos*** *(id_abono (PK), cliente (Fk → Clientes), saldo)*
3. ***Evento*** *(id_evento (PK), nombre_evento, fecha, asientos_disponibles)*
4. ***Reservas*** *(id_reserva (PK), cliente (FK → Clientes), evento (FK → Eventos),abono (FK → Abonos), fecha)*

Además, existen las siguientes secuencias:

Nombre de la secuencia	**Alimenta los valores de**
`seq_abonos`	La clave primaria de la tabla abonos: *id_abono*
`seq_evento`	La clave primaria de la tabla de eventos: *id_evento*
`seq_reservas`	La clave primaria de la tabla de reservas: *id_reserva*

Para facilitar la tarea se facilita el script **`reservaEvento_enun.sql`** junto a este enunciado, que además de crear las tablas previamente descritas contiene:

1. El procedimiento almacenado **`reset_seq(p_seq_name varchar)`** que sirve para resetear las secuencias, similar al utilizado en prácticas anteriores.
2. Un procedimiento almacenado **`inicializa_test`** que reinicia el contenido de la base de datos y la provee de unas filas de prueba (véase que `inicializa_test` llama a `reset_seq`).
3. Un procedimiento almacenado con los tests automáticos **`test_reserva_evento`** que llama a `inicializa_test` cada vez que prueba un caso.

En este trabajo **se pide la implementación de una transacción que permita reservar plaza en un evento del festival**.

```
create or replace procedure reservar_evento( arg_NIF_cliente
varchar, arg_nombre_evento varchar, arg_fecha date) is
```

Además, **se pide la implementación de la batería de test que comprueben la funcionalidad implementada.**

Para proceder a reservar un evento la transacción debe:

1. Comprobar que el evento no ha pasado. En caso contrario devolverá el **error -20001 con el mensaje '*No se pueden reservar eventos pasados*.'**

2. Si se cumple la comprobación anterior, y existe el evento para el que se solicita reserva, se comprobará que aún existen plazas/asientos para el evento y que el cliente tiene saldo en su abono. A lo largo de la transacción también pueden encontrarse errores que darán lugar a diferentes excepciones:

   - En caso de no existir dicho evento, debe lanzarse la **excepción -20003, con el mensaje '*El evento <nombre_evento> no existe*'**

   - En caso de que el cliente para el que se intenta hacer la reserva no exista, se lanzará **excepción -20002, con el mensaje de error 'Cliente inexistente'.**

   - En caso de que el cliente no disponga de saldo en su abono para reservar el evento, se lanzará la **excepción -20004, con el mensaje de error '*Saldo en abono insuficiente*'.**

3. Para formalizar la reserva, y habiéndose cumplido lo anterior, se deben realizar **tres acciones (y nunca una sin las otras):**

   - Descontar una unidad del saldo del abono del cliente.

   - Descontar en una unidad el número de plazas disponibles para el evento

   - Añadir la reserva a la tabla de reservas para el cliente, abono, evento y fecha correspondiente.

4. Incluye **en los comentarios del código** la respuesta a las siguientes preguntas (indicando la referencia a la pregunta en cada caso P4.1, P4.2, P4.3):

   P4.1 El resultado de la comprobación del paso 2 ¿sigue siendo fiable en el paso 3?:

   P4.2 En el paso 3, la ejecución concurrente del mismo procedimiento reservar_evento con, quizás otros o los mimos argumentos, ¿podría habernos añadido una reserva no recogida en esa SELECT que fuese incompatible con nuestra reserva?, ¿por qué?.

   P4.3. ¿Qué estrategia de programación has utilizado?

   P4.4. ¿Cómo puede verse este hecho en tu código?

   P4.5. ¿De qué otro modo crees que podrías resolver el problema propuesto? Incluye el pseudocódigo.

5. **BATERÍA DE TEST**: Comprobará al menos que se cumple que:

   T1. Si se intentar realizar una reserva con valores correctos, la reserva se realiza.

   T2. Si se intenta hacer una reserva de un evento pasado, devuelve error -20001 con el mensaje '*No se pueden reservar eventos pasados*.'

   T3. Si se intenta hacer una reserva a un evento inexistente devuelve el error -20003, con el mensaje 'El evento <nombre_evento> no existe'.

   T4. Si se intenta hacer una reserva a un cliente inexistente devuelve el error 20002, con el mensaje de error 'Cliente inexistente'.

T5. Si se intenta hacer una reserva para un cliente sin suficiente saldo en su abono, se lanzará el error -20004, con el mensaje de error 'Saldo en abono insuficiente'

2. Normas de Entrega

La práctica se realizará **EN GRUPOS DE TRES PERSONAS**

Formato de entrega:

- Se creará un **repositorio en github** para este trabajo, valorándose la contribución de cada persona del grupo en el resultado final, así como la progresividad en el avance (se penalizará entregas con commits sólo en el último momento. Por si no estáis muy familiarizad@s aún con git, aquí tenéis una guía básica de comandos.
- Para poder evaluarlo deberéis invitar a la profesora al repositorio (asmamolar@ubu.es) en el momento en el que lo creéis. En la cabecera del script que entregaréis, tenéis que añadir también el enlace al repositorio.
- Se enviará un fichero `.zip` a través de la plataforma **UBUVirtual** completando la tarea:

[ABD] Trabajo 2 - PLSQL-1C -23_24

El fichero comprimido (.zip o .rar) seguirá el siguiente formato de nombre, **sin utilizar tildes**:

Nombre_1erApellido_Nombre_1erApellido_Nombre_1erApellido.zip

- El fichero comprimido contendrá todos los scripts PL/SQL (formato .sql) generados para resolver el enunciado y que incluya todas las cuestiones planteadas.
- El **código entregado debe compilar** correctamente para poder ser evaluado, en caso contrario la evaluación será un 0.
- Se valorará la documentación del código. El **código debe estar bien documentado**, indicando las decisiones tomadas en cada caso, así como cualquier explicación que demuestre que se conocen los conceptos vistos en la asignatura.
- En caso de **alguna irregularidad** podría solicitarse una defensa del trabajo en horario de tutorías, a fijar con los profesores responsables de la asignatura. Una exagerada diferencia entre el resultado de este trabajo y el del examen del tema 2, suponen razón suficiente para solicitar esa defensa. En caso de no ser capaz de defender el trabajo, la calificación será modificada acorde.
- Fecha máxima de entrega **Jueves 11 de Abril a las 23.59h**.
- Aunque la entrega sea grupal, **deben entregarlo todas las personas que formen el grupo**. LA persona que no lo entregue en UBUVirtual en tiempo, no podrá ser evaluada.
- **NO SE PERMITEN ENTREGAS DESPUÉS DE LA FECHA LÍMITE.**