Práctica 6: Bloqueos y transacciones

Bases de datos

Objetivos

- Programar transacciones.
- Practicar el efecto de los bloqueos en lectura y escritura.
- Practicar el efecto de los niveles de aislamiento.

Enunciado

Se pide resolver los siguientes apartados y, como resultado de esta práctica, se debe subir al CV un documento PDF que describa la solución a cada uno de ellos.

En esta práctica necesitaremos usar SQL Developer y se arrancarán dos instancias del mismo programa, cada una representado a una transacción diferente.

Apartado 1. Bloqueos (Select)

1. Inicia una instancia de Oracle SQL Developer, a la que nos referiremos como **T1** en lo que sigue, y ejecuta lo siguente:

```
CREATE TABLE cuentas (
 numero number primary key,
    saldo number not null
   );

INSERT INTO cuentas VALUES (123, 400);
INSERT INTO cuentas VALUES (456, 300);
COMMIT;
```

2. Inicia otra instancia de SQL Developer a la que nos referiremos como **T2**.
3. Aunque es el comportamiento predeterminado, asegúrate de deshabilitar la autoconfirmación en ambas instancias con:

```
SET AUTOCOMMIT OFF
```

4. Desde T1 aumenta 100 euros el saldo de la cuenta 123.
5. Desde T2 consulta el saldo de la cuenta 123. ¿Cuánto es su saldo?
6. Desde T1 confirma los datos con:

```
COMMIT;
```

7. Desde T2 consulta el saldo de la cuenta 123. ¿Cuánto es su saldo?

Apartado 2. Bloqueos (Update)

Explica el comportamiento de las transacciones T1 y T2 a continuación (asegúrate de haber completado antes el apartado 1: los cambios de T1 deberían estar confirmados y el modo de autocompromiso desactivado en ambas):

1. Desde T1 aumenta en 100 euros el saldo de la cuenta 123.
2. Desde T2 aumenta 200 euros el saldo de la cuenta 123. ¿Se puede? ¿Qué le pasa a T2?
3. Desde T1 confirma los datos con:

```
COMMIT;
```

¿Qué le pasa a T2?

4. Desde T1 consulta el saldo de la cuenta 123. ¿Cuánto es su saldo?
5. Desde T2 confirmar los datos con:

   ```
   COMMIT;
   ```

6. Desde T1 consulta d nuevo el saldo de la cuenta 123. ¿Cuánto es su saldo?

Apartado 3. Bloqueos (Deadlock)

Explica el comportamiento de las transacciones T1 y T2 en los pasos a continuación (asegúrate de haber completado antes el apartado 2: los cambios de T1 y T2 deberían estar confirmados y el modo de autocompromiso desactivado en ambas). Si una transacción se queda bloqueada, continúa con el siguiente paso.

1. Desde T1 aumenta 100 euros el saldo de la cuenta 123.
2. Desde T2 aumenta 200 euros el saldo de la cuenta 456.
3. Desde T1 aumenta 300 euros el saldo de la cuenta 456.
4. Desde T2 aumenta 400 euros el saldo de la cuenta 123.
5. Desde T1 confima los cambios.

¿Qué ocurre?

Apartado 4. Niveles de aislamiento

Explica el comportamiento de las transacciones T1 y T2 a continuación (asegúrate de haber completado antes el apartado 3, confirma los últimos cambios si no lo has hecho ya, y mantén el modo de autocompromiso desactivado):

1. En T1:

   ```
   ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
   ```

2. En T1 :

   ```
   SELECT SUM(saldo) FROM cuentas;
   ```

3. En T2 :

   ```
   UPDATE cuentas SET saldo=saldo+100;
   COMMIT;
   ```

4. En T1:

   ```
   SELECT SUM(saldo) FROM cuentas;
   ```

   ¿Qué ha pasado?

5. En T1:

   ```
   ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
   ```

6. En T1 :

   ```
   SELECT SUM(saldo) FROM cuentas;
   ```

7. En T2 :

   ```
   UPDATE cuentas SET saldo = saldo+100;
   COMMIT;
   ```

8. En T1 :

   ```
   SELECT SUM(saldo) FROM cuentas;
   ```

¿Qué ha pasado? Explicar si hay alguna diferencia según los niveles de aislamiento.

Apartado 5. Transacciones

En una central de reservas de butacas para eventos (conciertos, circo, cine, …) se desea desarrollar una aplicación para que los clientes puedan reservar butacas en estos eventos. Se proporciona un script (**reservar.sql**, y que llama a otros dos: **preguntar.sql** y **no_preguntar.sql**) que consulta y actualiza los datos de dos tablas. La primera, **butacas(id number(8) primary key, evento nvarchar(30), fila nvarchar(10), columna nvarchar(10))** contiene todas las butacas disponibles de cada evento. La segunda, **reservas**, con el mismo esquema que **butacas**, almacena las butacas que se han reservado. El script incluye los valores para el evento, fila y columna, comprueba que la butaca exista y consulta si no está reservada previamente.

Al ejecutarlo, y si se encuentra disponible la butaca, pide la confirmación para la reserva (el usuario debe responder con **s** o **n**) y después inserta en la tabla **reservas** una tupla que representa a la butaca reservada. La variable **v_confirmar** puede contener estos valores '**s**' o '**n**' (según el usuario quiera confirmar realizar la reserva o no, respectivamente). Mientras el script **preguntar.sql** pide el valor de esta variable por consola, el script **no_preguntar.sql** proporciona directamente su valor: '**n**', que corresponde a una situación en que no se puede realizar la reserva. Este último script se ejecutará solo si hay un error en la reserva (es decir, ya existe una reserva para la misma butaca o no existe la butaca). Mira las notas al final que comentan algunas de las instrucciones usadas en el script.

1. Cierra las instancias anteriores de SQL Developer y abre una nueva (con el nivel de aislamiento y autocompromiso predeterminados).
2. Crea las tablas y secuencias para los identificadores:

```
CREATE TABLE butacas(id number(8) primary key,
                     evento  varchar(30),
                     fila    varchar(10),
                     columna varchar(10)) ;

CREATE TABLE reservas(id number(8) primary key,
                     evento  varchar(30),
                     fila    varchar(10),
                     columna varchar(10)) ;

CREATE SEQUENCE Seq_Butacas INCREMENT BY 1 START WITH 1 NOMAXVALUE;

CREATE SEQUENCE Seq_Reservas INCREMENT BY 1 START WITH 1 NOMAXVALUE;
```

3. Inserta algunos valores de prueba en butacas:

```
INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','1');

INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','2');

INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','3');

COMMIT;
```

4. Modifica en el script **reservar.sql** las rutas que hacen referencia a los otros dos scripts **preguntar.sql** y **no_preguntar.sql** según la carpeta en donde los hayas descargado. Con **reservar.sql** se va a reservar la butaca de la fila 1, columna 1 para **'Circo'**. Estos datos se encuentran definidos en las siguientes constantes:

```
DEF v_evento='Circo';

DEF v_fila='1';

DEF v_columna='1';
```

   Si examinas el script, la variable **v_error** está pensada para identificar el hecho de que no se haya encontrado una butaca para la reserva deseada (en este caso **v_error** será **true** y, en caso contrario, **false**). Si encuentra la butaca libre escribirá "**INFO: Se intenta reservar.**". Dado que la tabla **reservas** está vacía, se debería poder reservar, indicando "**INFO: Localidad reservada**. "

   Ejecuta el script con: **@ *ruta*\reservar.sql** para realizar la reserva, donde ***ruta*** es la carpeta en la que hayas descomprimido los scripts. En la salida del script aparecerá '**¿Confirmar la reserva?**', y se responderá '**s**' para confirmarla. Observa el resultado por consola (valores de las variables que se han mostrado en pantalla) y examina el contenido de la tabla **reservas**.

5. Intenta reservar de nuevo la misma fila desde la misma instancia de SQL Developer y comprueba que no sea posible.

6. Realiza una nueva reserva desde la misma instancia para la butaca de la fila 1, columna 4 para **'Circo'** y comprueba que no es posible porque no existe esa butaca.
7. Realiza una nueva reserva desde la misma consola para la butaca de la fila 1, columna 2 para **'Circo'** pero sin realizar aún la confirmación (no introduzcas todavía '**s**' cuando se te pregunte).
8. Abre una nueva instancia de SQL Developer y realiza la misma reserva anterior desde esta instancia. Confirma la reserva introduciendo '**s**'.
9. Confirma la reserva del punto 6 introduciendo '**s**'. ¿Qué sucede?
10. ¿Cómo se podría solucionar el problema del punto anterior sin perjudicar el grado de concurrencia?

NOTAS:

- **SET DEFINE ON**: Activa la sustitución de variables, estableciendo el caracter predefinido que las identifica (&).
- **DEF *variable = valor***: Define una variable de sustitución (constante) para el script.
- **SET ECHO OFF**: Desactiva la salida de comandos en un script.
- **SET VERIFY OFF**: Desactiva la visualización automática de las variables de sustitución (evita una salida verbosa).
- **VARIABLE *nombre tipo***: Define una variable de scripting con un tipo asociado.
- **COLUMN SCRIPT_COL NEW_VALUE SCRIPT**: Define una variable de sustitución denominada **SCRIPT_COL** cuyo valor se consulta haciendo referencia a **SCRIPT**. Permite usar la variable en los scripts SQL*Plus como si fueran variables de PL/SQL. Se usa en la siguiente instrucción:

  ```
  SELECT DECODE(TRIM(:v_error), 'false', '"C:\tmp\preguntar.sql"', '"C:\tmp\no_preguntar.sql"') AS SCRIPT_COL from dual;
  ```

  para obtener en **SCRIPT_COL** el valor del script a ejecutar más adelante (ya sea "**preguntar.sql**" o "**no_preguntar.sql**". Recuerda que hay que modificar las rutas para ajustarlas a donde estén estos scripts.
- **PROMPT *texto***: Muestra el texto que se indica a continuación (también variables de sustitución). Generalmente viene acompañado de un **ACCEPT** para entrada de datos.
- **PRINT *variable***: Muestra el valor de la variable de scripting.
- **@ &SCRIPT**: La arroba se usa para ejecutar un script como ya hemos visto. En este caso el script se proporciona mediante la variable de sustitución **SCRIPT**.