Práctica 6: Bloqueos y transacciones

Bases de datos

Objetivos

- Programar de 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:

Apartado 1. Bloqueos (Select)

1. Inicia Oracle SQLDeveloper (a la que denominaremos T1) 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 SQLDeveloper (a la que denominaremos T2).
3. 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 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 consultar el saldo de la cuenta 123. ¿Cuánto es su saldo?

Apartado 3. Bloqueos (Deadlock)

Explica el comportamiento de las transacciones T1 y T2 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):

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.

¿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 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 (**`script.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 existe y consulta si no está reservada previamente. Al ejecutarlo, pide la

confirmación para la reserva y después inserta en la tabla **reservas** una tupla que representa a la butaca reservada.

1. 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;
```

2. 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;
```

3. Prueba el script desde SQLDeveloper reservando la fila 1, columna 1 para **'Circo'**. Antes hay que modificar en este script las rutas que hagan referencia a otros según la carpeta en donde se hayan depositado.
4. Intenta reservar de nuevo la misma fila desde la misma instancia de SQL Developer y comprueba que no sea posible.
5. Realiza una nueva reserva desde la misma instancia para la fila 1, columna 4 para **'Circo'** y comprueba que no es posible porque no existe esa butaca.
6. Realiza una nueva reserva desde la misma consola para la fila 1, columna 2 para **'Circo'** pero sin realizar aún la confirmación.
7. Abre una nueva instancia de SQL Developer y realiza la misma reserva anterior desde esta instancia. Confirma la reserva.
8. Confirma la reserva del punto 6. ¿Qué sucede?
9. Modifica el script para resolver el punto anterior.