UA04. GESTIÓN DE LA INFORMACIÓN

4.10. Transacciones

Cuando tenemos una serie de consultas SQL que deben ejecutarse en conjunto, con el uso de **transacciones** podemos asegurarnos de que nunca nos quedaremos a medio camino de su ejecución.

Las transacciones tienen la característica de poder "**deshacer**" los cambios efectuados en las tablas, de una transacción dada, si no se han podido realizar todas las operaciones que forman parte de dicha transacción.

Por eso, las bases de datos que soportan transacciones son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor que almacena la base de datos, ya que las consultas se ejecutan o no en su totalidad.

Al ejecutar una transacción, el motor de base de datos garantiza: **atomicidad, consistencia, aislamiento y durabilidad (ACID)** de la transacción (o conjunto de comandos) que se utilice.

El ejemplo típico que se pone para hacer más clara la necesidad de transacciones en algunos casos es el de una transacción bancaria:

Para transferir una cantidad de dinero de la cuenta de Antonio a la cuenta de Pedro, se necesitarían dos consultas:

1. En la cuenta de Antonio para quitar de su cuenta ese dinero.
2. En la cuenta de Pedro para añadir ese dinero a su cuenta.

```
1 UPDATE cuentas SET saldo = saldo - cantidad WHERE cliente = "Ant
```

```
1 UPDATE cuentas SET saldo = saldo + cantidad WHERE cliente = “Pec
```

Pero, ¿qué ocurre si por algún imprevisto (un apagón de luz, etc.), el sistema “cae” después de que se ejecute la primera consulta, y antes de que se ejecute la segunda? Antonio tendrá una cantidad de dinero menos en su cuenta y creerá que ha realizado la transferencia. Pedro, sin embargo, creerá que todavía no le han realizado la transferencia.

Commit y Rollback

Una transacción tiene dos finales posibles, **COMMIT** o **ROLLBACK**. Si se finaliza correctamente y sin problemas se hará con **COMMIT**, con lo que los cambios se realizan en la base de datos, y si por alguna razón hay un fallo, se deshacen los cambios efectuados hasta ese momento, con la ejecución de **ROLLBACK**.

Por defecto, al menos en **MySQL**, en una conexión trabajamos en modo **autocommit** con valor **true**. Eso significa que cada consulta es una transacción en la base de datos.

Por tanto, si queremos definir una transacción de varias operaciones, estableceremos el modo **autocommit** a **false** con el método **setAutoCommit** de la clase **Connection**.

En modo **no autocommit** las transacciones quedan definidas por las ejecuciones de los métodos **commit** y **rollback**. Una transacción abarca desde el último commit o rollback hasta el siguiente commit. Los métodos **commit** o **rollback** forman parte de la clase **Connection**.

En la siguiente porción de código de un procedimiento almacenado, puedes ver un ejemplo sencillo de cómo se puede utilizar **commit** y **rollback**: tras las operaciones se realiza el commit, y si ocurre una excepción, al capturarla realizaríamos el rollback.

```
1
2 BEGIN
3 …
```

```
SET AUTOCOMMIT OFF
update cuenta set saldo=saldo + 250 where dni=”12345678-L”;
update cuenta set saldo=saldo - 250 where dni=”89009999-L”;
COMMIT;
…
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK ;
END;
```

Es conveniente planificar bien la aplicación para minimizar el tiempo en el que se tengan transacciones abiertas ejecutándose, ya que **consumen recursos** y suponen bloqueos en la base de datos que puede parar otras transacciones. En muchos casos, un diseño cuidadoso puede evitar usos innecesarios que se salgan fuera del modo estándar **AutoCommit**.

Ejemplo en Java

```
public static void transaccion() throws SQLException {
    con.setAutoCommit(false);   // Se desactiva el AutoCommit pa
    Statement st = con.createStatement();
    String sql;
    sql="drop table if exists cliente;";
    st.execute(sql);    // Se elimina la tabla si existiera
    System.out.println("Elimina la tabla");
    sql="CREATE TABLE cliente(id int primary key, nombre varchar
    st.execute(sql);    // Se crea la tabla
    System.out.println("Crea la tabla");
    sql="insert into cliente VALUES (1,'UNO');";
    st.executeUpdate(sql);  // Inserta el registro 1
    System.out.println("Inserta registro 1");
    try{
```

```
        con.commit();       // Comienza la transacción
        sql="insert into cliente VALUES (2,'DOS');";
        st.executeUpdate(sql);  // Inserta el registro 2
        System.out.println("Inserta registro 2");
        sql="insert into cliente VALUES (3,'TRES');";
        st.executeUpdate(sql);  // Inserta el registro 3
        System.out.println("Inserta registro 3");
        sql="insert into cliente VALUES (3,'CUATRO');";
        st.executeUpdate(sql);  // Intenta insertar el registro
        System.out.println("No inserta registro al exister el II
    }catch(SQLException e) {
        con.rollback();     // Deshace las dos últimas insercior
    }
    con.setAutoCommit(true);    // Se vuelve a activar el AutoCc
}
```

```

public static void transaccion() throws SQLException {
        con.setAutoCommit(false);   // Se desactiva el AutoCommi
        String sql;
        sql="drop table if exists cliente;";
        PreparedStatement st = con.prepareStatement(sql);
        st.execute();   // Se elimina la tabla si existiera
        System.out.println("Elimina la tabla");
        sql="CREATE TABLE cliente(id int primary key, nombre var
        st.execute(sql);    // Se crea la tabla
        System.out.println("Crea la tabla");

```

```
12          sql="insert into cliente VALUES (?,?);";
13          st=con.prepareStatement(sql);
14          st.setInt(1, 1);
15          st.setString(2, "UNO");// Inserta el registro (1,"UNO")
16          st.executeUpdate();
17          System.out.println("Inserta registro 1");
18          try{
19              con.commit();       // Comienza la transacción
20              st.setInt(1, 2);
21              st.setString(2, "DOS");// Inserta el registro (1,"UN
22              st.executeUpdate();
23              System.out.println("Inserta registro 2");
24              st.setInt(1, 3);
25              st.setString(2, "TRES");// Inserta el registro (1,"t
26              st.executeUpdate();
27              System.out.println("Inserta registro 3");
28              st.setInt(1, 3);
29              st.setString(2, "CUATRO");// Inserta el registro (1,
30              st.executeUpdate();
31              System.out.println("Inserta registro 4");
32          }catch(SQLException e) {
33              con.rollback();     // Deshace las dos últimas inser
34          }
35          con.setAutoCommit(true);    // Se vuelve a activar el Au
        }
```