MP_0486. Acceso a datos
UF2. Manejo de conectores a bases de datos relacionales

2.4. Gestión de transacciones

Índice

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Conocer el concepto de transacción.
- Realizar programas Java con el API JDBC que agrupen un conjunto de operaciones de actualización de la base de datos dentro de una transacción.

Transacciones

Concepto

Una transacción es el conjunto de operaciones necesarias para llevar a cabo una determinada tarea que implique modificaciones en la base de datos.

Por ejemplo, para realizar una venta serán necesarias las siguientes operaciones en la base de datos:

- Añadir un nuevo registro con los datos del cliente, si no está registrado en el sistema.
- Añadir la nueva factura.
- Añadir las líneas de detalle de la venta.
- Actualizar el stock en el almacén.

Todas estas operaciones **deben ser tratadas dentro de una unidad de trabajo, de modo que se realicen todas las operaciones o ninguna**. Sería un desastre que la tarea se quedara a medias por un error; por ejemplo, que se añadiera la factura y no se actualizara el stock.

Para garantizar que esto no ocurra, todo el bloque de operaciones ha de estar incluido dentro de una transacción.

Propiedades ACID

Una transacción debe cumplir con un grupo de propiedades denominadas "propiedades ACID".

Las siglas ACID corresponden a las iniciales de:

Atomicidad

La transacción debe ser completa, es decir, **se ejecutarán todas las operaciones o ninguna**.

Consistencia

La ejecución de la transacción **no debe romper las reglas de integridad**. Por ejemplo: la transacción no debe provocar que se agregue una factura sin cliente relacionado, o una línea de detalle que no pertenezca a ninguna factura.

Aislamiento

La ejecución de **una transacción no debe afectar al funcionamiento de otra transacción**. El SGBD debe garantizar una buena gestión de bloqueo de registros cuando sea necesario.

Durabilidad

Es decir, la **persistencia (permanencia en el tiempo) de los nuevos datos.**

Un SGBD que quiera llamarse transaccional debe cumplir las cuatro propiedades ACID obligatoriamente .

Control de transacciones en MySQL

La gestión de transacciones en MySQL se realiza a partir de estas tres sentencias:

BEGIN

Inicio de la transacción.

COMMIT

Finalización de la transacción.

ROLLBACK

Aborta la transacción revirtiendo los cambios.

Para agrupar un conjunto de operaciones dentro de una transacción hay que encerrarlas en un **bloque BEGIN ... COMMIT.**

```
USE FERRETERIA;

BEGIN;
   INSERT INTO CLIENTE VALUES('12345678A','LUIS LOPEZ LOPEZ', 'C/ SOL, 3','639-
639-639', 'MADRID');
   INSERT INTO FACTURA VALUES(5446,'2017-12-15',false,'12345678A');
   INSERT INTO DETALLE VALUES(5446,'MAR1',1,13);
   INSERT INTO DETALLE VALUES(5446,'TOR7',2,0.9);
   UPDATE PRODUCTO SET STOCK=STOCK-1 WHERE CODIGO='MAR1';
   UPDATE PRODUCTO SET STOCK=STOCK-2 WHERE CODIGO='TOR7';
COMMIT;
```

Ahora vamos a realizar una transacción similar para otro cliente y otra factura. Pero esta vez la abortaremos en el último momento.

```
BEGIN;
   INSERT INTO CLIENTE VALUES('23456789A','ROSA LOPEZ LOPEZ', 'C/ LUNA, 5','612-
612-612', 'MADRID');
   INSERT INTO FACTURA VALUES(5447,'2017-12-15',false,'23456789A');
   INSERT INTO DETALLE VALUES(5447,'MAR1',1,13);
   INSERT INTO DETALLE VALUES(5447,'TOR7',2,0.9);
   UPDATE PRODUCTO SET STOCK=STOCK-1 WHERE CODIGO='MAR1';
   UPDATE PRODCTO SET STOCK=STOCK-2 WHERE CODIGO='TOR7';
   SELECT * FROM DETALLE;
ROLLBACK;
```

Dentro del código de la transacción hemos colocado una sentencia SELECT * FROM DETALLE. Verás que la sentencia ha devuelto todas las líneas de detalle, incluidas las de la nueva factura (5447). Ahora, ejecuta de nuevo la siguiente sentencia:

```
SELECT * FROM DETALLE;
```

Verás que ahora no aparecen los detalles de la nueva factura, porque después del ROLLBACK los cambios han sido revertidos.

IMPORTANTE: si inicias una transacción (BEGIN) y luego no haces el COMMIT, la transacción terminará siendo rechazada, como si hubieras hecho un ROLLBACK.

Estados de una transacción

Las transacciones pueden encontrarse en uno de los siguientes estados:

- **Activa:** la transacción está activa durante todo el tiempo que dura su ejecución.
- **Parcialmente comprometida:** se acaba de realizar la última instrucción de la transacción, pero todavía no se ha hecho COMMIT TRAN (la finalización de la transacción).
- **Fallida:** ha ocurrido algún fallo en alguna de las operaciones de la transacción y la transacción no puede finalizar. Después de pasar por este estado, pasará por el estado de *abortada*.
- **Abortada:** la transacción pasa a este estado cuando se ha restablecido la BD a su estado anterior. En este caso, se deshacen los cambios realizados como si no hubiese ocurrido nada.
- **Comprometida:** cuando todas las operaciones se han completado con éxito y la BD ha quedado actualizada.

Transacciones en Java

Aplicación práctica

En este apartado pondremos en práctica el uso de transacciones desde un programa Java. El siguiente programa dará de alta una nueva factura en la base de datos FERRETERIA. Dicha tarea implica **dar de alta al cliente, añadir la factura, añadir las líneas de detalle y actualizar el stock de los productos vendidos.**

Para empezar, crea un nuevo proyecto en eclipse, y copia y pega este código para la clase *Principal*.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class Principal {

	public static void main(String[] args) throws SQLException {

		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para
MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/Ferreteria";
		String user = "root";
		String pass = "amelia";
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user,
pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la
conexión con la BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de
datos");

		// Paso 3: Interactuar con la BD
		try {
			con.setAutoCommit(false);
			Statement sentencia = con.createStatement();
```

```
                        String sql = "INSERT INTO CLIENTE VALUES
('51666666A','ROCAFLOR DELGADO RODOLFO', 'C/ PITONISA, 45', '616656644',
'SEVILLA');";
                        sentencia.executeUpdate(sql);
                        sql = "INSERT INTO FACTURA VALUES (5446,'2018/04/23', 0,
'51666666A');";
                        sentencia.executeUpdate(sql);
                        sql = "INSERT INTO DETALLE VALUES (5446,'MAR2', 1, 7);";
                        sentencia.executeUpdate(sql);
                        sql = "INSERT INTO DETALLE VALUES (5446,'TOR7', 2,
0.8);";
                        sentencia.executeUpdate(sql);
                        sql = "UPDATE PRODUCTO SET STOCK=STOCK-1 WHERE
CODIGO='MAR2'";
                        sentencia.executeUpdate(sql);
                        sql = "UPDATE PRODUCTO SET STOCK=STOCKKK-2 WHERE
CODIGO='TOR7'";
                        sentencia.executeUpdate(sql);

                        con.commit();

                } catch (SQLException e) {
                        System.out.println("Ha ocurrido un error al añadir la
factura");
                        System.out.println(e.getMessage());
                        con.rollback();
                }


                // Paso 4: Cerrar la conexión
                try {
                        con.close();
                } catch (SQLException e) {
                        System.out.println("No se ha podido cerrar la conexión
con la BD");
                        System.out.println(e.getMessage());
                        return;
                }
                System.out.println("Se ha cerrado la base de datos");
        }

}
```

Este programa ejecuta seis sentencias de manipulación de datos, cuatro inserciones y dos actualizaciones. Sin embargo, observa que en la última sentencia SQL la palabra *STOCK*, está mal escrita, provocando así una excepción que llevará el control del programa al bloque *catch*, donde se aborta la transacción con un método *rollback()*.

En realidad, ninguna modificación ha sido efectuada a pesar de que las cinco primeras sentencias fueron correctas. **La gestión de la transacción está garantizando que se ejecuten todas las sentencias o ninguna.**

IMPORTANTE: no te olvides de importar el driver de MySQL para Java.

Ahora, vamos a analizar más detenidamente el código anterior:

con.setAutoCommit(false);

El método *setAutoCommit()* de la clase *Connection* permite establecer el valor de la propiedad *autoCommit* para especificar cuándo será completada una transacción. Esta propiedad tiene por defecto el valor *true*, lo que significa que cada instrucción SQL se ejecuta y confirma en una transacción individual. Por lo tanto, mientras la propiedad *autoCommit* tenga el valor *true*, no será posible agrupar varias sentencias SQL en una única transacción.

Por ello, necesitamos comenzar la transacción estableciendo el valor *false* a la propiedad *autoCommit*.

con.commit();

El método *commit()* finaliza la transacción, realizando físicamente todas las actualizaciones especificadas por la colección de sentencias SQL que se han ido ejecutando.

con.rollback();

Aborta la transacción revirtiendo todos los cambios efectuados por el grupo de sentencias SQL que se han ejecutado dentro de la transacción. Se utiliza el método *rollback()* como respuesta a una situación de excepción.

Si has ejecutado el programa, habrás comprobado que la transacción ha funcionado como esperábamos y ninguna de las modificaciones se ha llegado a efectuar.

Ahora, prueba a corregir la palabra *STOCK* en la última sentencia SQL para que puedas comprobar después si se efectuaron todas las modificaciones, añadiéndose la nueva factura.

Tendrás que utilizar el entorno de MySQL Workbench para comprobar que se han guardado todos los datos.

Despedida

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Una **transacción** es el conjunto de operaciones necesarias para llevar a cabo una determinada tarea que implica modificaciones en la base de datos. Un SGBD, para llamarse transaccional, debe garantizar las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad) en las transacciones.
- Java trata cada instrucción SQL dentro de una transacción individual de manera predeterminada. Para agrupar varias instrucciones en una sola transacción hay que comenzar por cambiar dicho comportamiento, ejecutando la siguiente sentencia:

  con.setAutoCommit(false);

  Siendo *con* el objeto *Connection*.

- El método *commit()* de la clase *Connection* finaliza la transacción, permitiendo así la actualización de la base de datos con todos los cambios especificados por las sentencias SQL ejecutadas.
- El método *rollback()* de la clase *Connection* aborta la transacción.