1.3. Manipulación de información

Llamamos manipulación de información a las operaciones de:

- Inserción
- Modificación
- Eliminación

de filas.

Las tres operaciones las vamos a realizar de la misma manera a través de Java y, como ocurre con las consultas de información se podrán realizar tanto de forma parametrizada como no parametrizada.

1.3.1. Manipulación de información de forma no parametrizada

La manipulación de información de forma no parametrizada se realiza de la misma forma que la consulta, con la única diferencia de que en lugar de utilizar el método ResultSet executeQuery(consulta) para solicitar la operación, lo vamos a hacer con el método int execueteUpdate(consulta). El valor que nos va a devolver este método es el número de filas que han sido afectadas.

```
int execueteUpdate(consulta)
```

Veamos un ejemplo sencillo.

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

public class UT12 {

    public static void main(String[] args) {

        String cadenaConexion =
"jdbc:mysql://localhost:3306/pruebasprogramacion";
```

```
        String consulta = "UPDATE Tabla1 SET nombre = 'Paco' WHERE id =
10";
        Connection conexion = null;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = DriverManager.getConnection(cadenaConexion,
"root", "");
            Statement state = conexion.createStatement();
            int result = state.executeUpdate(consulta);
            System.out.println("Se han actualizado " + result +
"filas")

        } catch (Exception e) {
            System.out.print("Ha ocurrido un error al intentar
conectarme: ")
            System.out.println(e.toString());
        } finally{
            try {
                if(conexion != null){
                    conexion.close();
                }
            } catch (Exception e2) {
                System.out.println(e2.toString());
            }
        }

    }

}
```

Si lo que necesitamos es realizar un INSERT o un DELETE, se realizarían de la misma forma, pero cambiando la consulta.

1.3.2. Manipulación de información de forma parametrizada

Aquí nos pasa exactamente lo mismo que en el caso de la forma no parametrizada, solo tenemos que cambiar executeQuery() por executeUpdate().

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.PreparedStatement;

public class UT12 {

    public static void main(String[] args) {

        String cadenaConexion =
"jdbc:mysql://localhost:3306/pruebasprogramacion";
        String consulta = "UPDATE Tabla1 SET nombre = ? WHERE id = ?";
        Connection conexion = null;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = DriverManager.getConnection(cadenaConexion,
"root", "");
            PreparedStatement ps = conexion.prepareStatement(consulta);
            ps.setString(1, "Paco");
            ps.setInt(2, 10);
            int result = ps.executeUpdate();

            System.out.println("Se han actualizado " + result +
"filas")

        } catch (Exception e) {
            System.out.println(e.toString());
        } finally{
            try {
                if(conexion != null){
                    conexion.close();
                }
            } catch (Exception e2) {
                System.out.println(e2.toString());
            }
        }

    }

}
```

1.3.3. Utilización de transacciones con la manipulación de información

Debemos acostumbrarnos a utilizar transacciones siempre que vayamos a realizar manipulación de datos en una base de datos, incluso cuando lo hacemos desde un programa que accede a la misma.

Java nos proporciona una serie de métodos que nos permiten crear una nueva transacción y luego realizar el commit o el rollback, según sea necesario.

Estos métodos son del objeto conexión:

- setAutoCommit(false): con este método deshabilitamos el sistema autocommit y, por tanto, nuestras operaciones no se aplicarán a la base de datos hasta que no hagamos un commit de forma explícita.
- commit(): aplica las operaciones que hayamos realizado sobre la base datos, después de poner el autocommit a falso. La utilizaremos en el caso de que no se haya producido ningún error.
- rollback(): descarta todas las operaciones sobre la base de datos, después de poner el autocommit a falso. La utilizaremos en el caso de que se haya producido algún error.

Veamos un ejemplo:

```
public class UT12 {

    public static void main(String[] args) {

        String cadenaConexion =
"jdbc:mysql://localhost:3306/pruebasprogramacion";
        String consulta = "UPDATE Tabla1 SET nombre = ? WHERE id = ?";
        Connection conexion = null;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = DriverManager.getConnection(cadenaConexion,
"root", "");
            conexion.setAutoCommit(false);

            PreparedStatement ps = conexion.prepareStatement(consulta);
            ps.setString(1, "Paco");
            ps.setInt(2, 10);
            int result = ps.executeUpdate();
```

```
            conexion.commit();
            System.out.println("Se han actualizado " + result +
"filas")
        } catch (Exception e) {
            System.out.println(e.toString());
            try{
                if(conexion != null)
                    conexion.rollback();
            }catch(Exception e1){
                System.out.println(e1.toString());
                }
        } finally{
            try {
                if(conexion != null){
                    conexion.close();
                }
            } catch (Exception e2) {
                System.out.println(e2.toString());
            }
        }

    }

}
```

Gracias al uso de transacciones podremos proteger la integridad y la estabilidad de la información de la base de datos.

Ahora probemos a realizar nosotros mismos estas operaciones…