Acceso a bases de datos relacionales desde Java con MySQL Workbench

Prefacio

La forma de acceso a la base de datos MySQL Workbench desde Java ha variado, puesto que el conector actualizado tiene una forma distinta de registrar los drivers, además de que hay que poner algo en el mismo para configurar correctamente la zona horaria (O bien arreglarlo desde el servidor):

Tenemos, por ejemplo,

Clase Coche

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ficherobd;

/**
 *
 * @author
 */
public class Coche
{
    private String matricula, marca, modelo, color;
    private int año;
    private double precio;

    public Coche(String matricula, String marca, String modelo, String color, int año, double precio) {
        this.matricula = matricula;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
        this.año = año;
        this.precio = precio;
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }
```

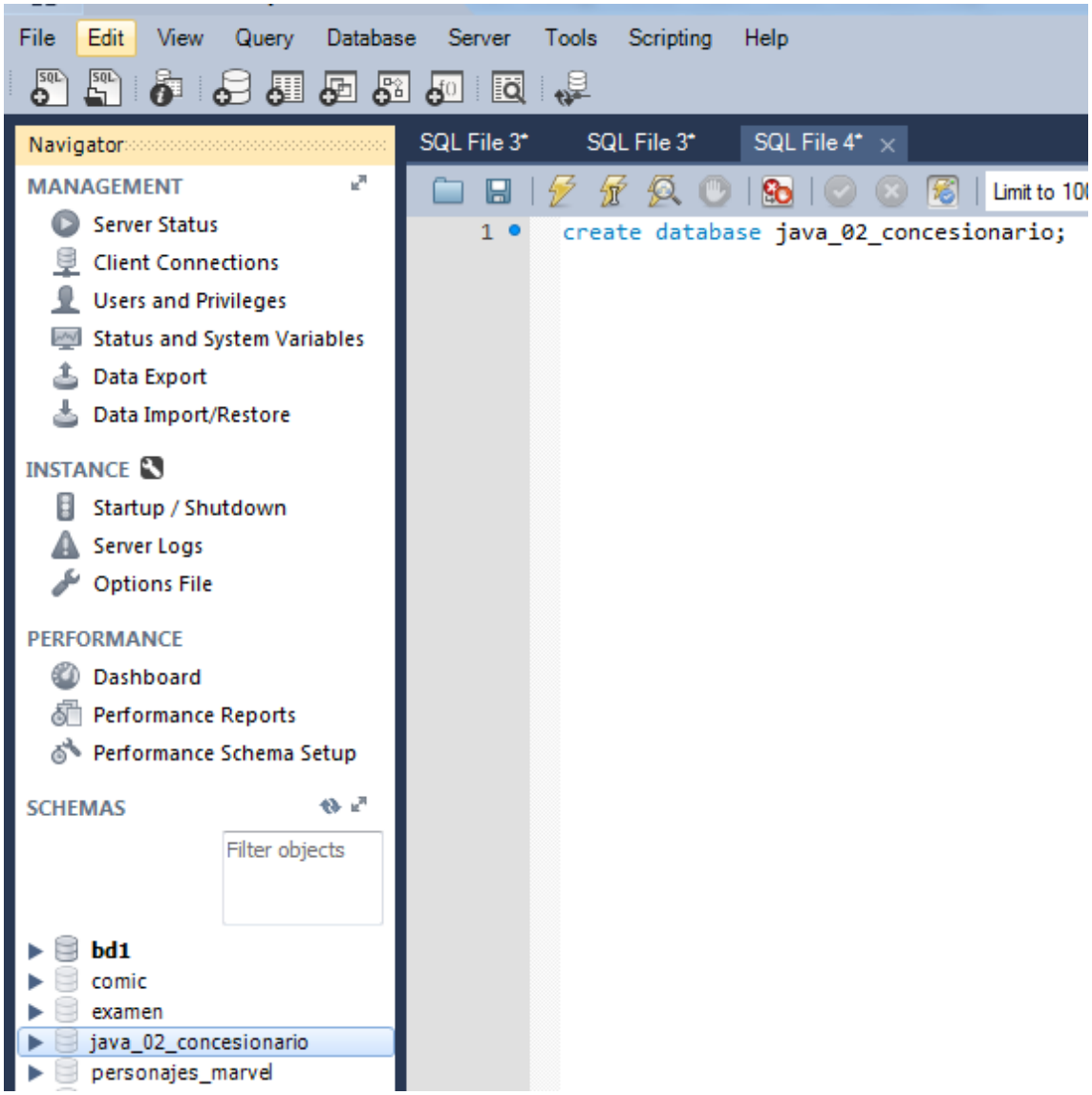
```
46
47      public void setModelo(String modelo) {
48          this.modelo = modelo;
49      }
50
51      public String getColor() {
52          return color;
53      }
54
55      public void setColor(String color) {
56          this.color = color;
57      }
58
59      public int getAño() {
60          return año;
61      }
62
63      public void setAño(int año) {
64          this.año = año;
65      }
66
67      public double getPrecio() {
68          return precio;
69      }
70
71      public void setPrecio(double precio) {
72          this.precio = precio;
73      }
74
75      @Override
        public String toString() {
77          return "Coche{" + "matricula=" + matricula + ", marca=" + marca + ", modelo=" + modelo +
                    ", color=" + color + ", a\u00f1o=" + año + ", precio=" + precio + '}';
79      }
80
81
82
83
84
85  }
86
```

Ésta es una clase con seis atributos, sus correspondientes setters y getters y un método toString que sobreescribe el método toString por defecto, mostrándonos toda la información de los objetos de la clase que se instancien.

Clase BaseDatos

Se utilizará para crear una nueva instancia de la base de datos (Se sobreentiende que la contraseña de root es “usuario” en el MySQL Workbench)

Lógicamente, debe estar creada la base de datos en el servidor de bases de datos:



```java
package ficherobd;

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

/**
 *
 * @author
 */
public class BaseDatos
{
    static Connection conn;
    static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    static final String USER = "root";
    static final String PASS = "Contraseña";
    static final String BD="java_02_concesionario";
    static final String DB_URL = "jdbc:mysql://localhost:3306/"+BD+"?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDate
    private static BaseDatos INSTANCE;

    /**
     * Patrón de diseño singleton
     */
    private BaseDatos()
    {
        try{
            Class.forName(JDBC_DRIVER).newInstance();
            conn=DriverManager.getConnection(DB_URL,USER,PASS);
            if(conn!=null){
                System.out.println("Conexión a la base de datos "+DB_URL+".....CORRECTA");
            }
        }
        catch(SQLException | InstantiationException | IllegalAccessException ex){
            System.err.println("Problemas al conectar"+ex.getMessage());
        }
        catch(ClassNotFoundException ex){
            System.err.println(ex.toString());
        }
    }

    public static BaseDatos getInstance()
    {
        if(INSTANCE == null)
            INSTANCE = new BaseDatos();
        return INSTANCE;
    }


    public Connection getConnection()
    {
        return conn;
    }



}
```

Los atributos de la clase BaseDatos (Todos ellos son estáticos)

Un atributo de tipo connection, conn

Un atributo final de tipo String con el Driver que vamos a utilizar en el programa, que se incluye en el siguiente conector:

Un atributo final de tipo String en el que se almacenará el usuario con el que se va a acceder a la base de datos

Un atributo final de tipo String en el que se almacenará la contraseña correspondiente al usuario con el que se va a enlazar a la base de datos.

Un atributo final de tipo String en el que se almacenará el nombre de la base de datos a la que queremos conectarnos.

Un atributo final de tipo String en el que se almacenará la dirección url para acceder a la base de datos. Además, si se quiere acceder a una base de datos en MysQL Workbench desde Netbeans, debe usarse

+"?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";

Después de la dirección url de la base de datos.

NOTA: Si no se ha configurado correctamente el driver, mediante la línea

static final String DB_URL = "jdbc:mysql://localhost:3306/"+BD+"?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";

el programa nos saca lo siguiente por pantalla:

(También puede arreglarse modificando el archivo my.ini en el MySQl Workbench)

El último atributo es un objeto privado del tipo BaseDatos (Sólo se declara, no se instancia)

El método constructor BaseDatos() nos permite crear una instancia nueva del driver y conseguir una conexión mediante las líneas:

```
Class.forName(JDBC_DRIVER).newInstance();
        conn=DriverManager.getConnection(DB_URL,USER,PASS);
```

donde conn es el atributo de tipo connection

(Lógicamente englobadas dentro de un try-catch para control de las posibles excepciones)

Los dos últimos métodos son:

getInstance, de tipo BaseDatos, público y estático, que lo que hace es crear una instancia de BaseDatos si esta no existe ya e igualar la instancia creada al atributo de tipo BaseDatos. Retorna esa instancia para poder trabajar con sus métodos

getConnection, de tipo Connection, público y estático, que me devuelve la conexión creada en el atributo con.

MetodosBBDD.java

```java
package ficherobd;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author
 */
public class MetodosBBDD
{
    public static void crearTablaCoches()
    {
        Connection conexion = BaseDatos.getInstance().getConnection();
        final String sql = "CREATE TABLE IF NOT EXISTS coches "
                + "(matricula varchar(8),marca varchar(40),modelo varchar(40),"
                + "color varchar(40), año int not null,"
                + "precio decimal not null,"
                + "PRIMARY KEY (matricula))";
        try
        {
            Statement sentencia = conexion.createStatement();
            sentencia.executeUpdate(sql);

        }
        catch (SQLException ex)
        {
            System.out.println("Error al crear la tabla");
            System.err.println(ex.getMessage());
        }
    }


```

Esta clase está compuesta en su totalidad por métodos estáticos,

En el método crearTablacoches(), la línea

Connection conexion = BaseDatos.getInstance().getConnection();

Me crea una instancia de BaseDatos y me devuelve su conexión, si todo va bien

Después se almacena una consulta que permite la tabla coches si no existe y en el try, se ejecuta la consulta mediante las órdenes:

Statement sentencia = conexion.createStatement();

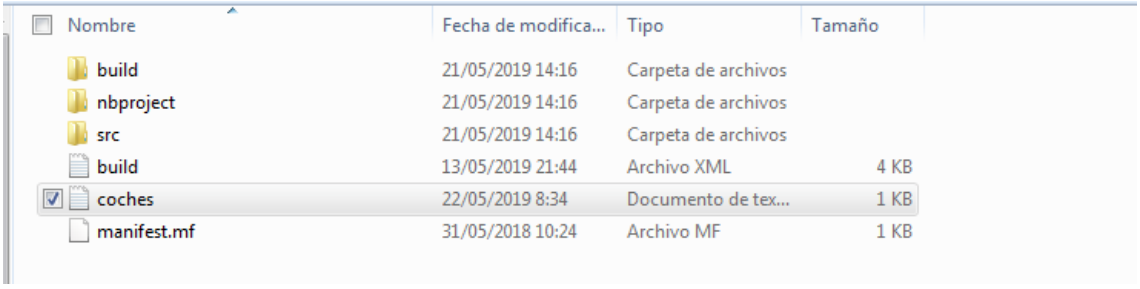
sentencia.executeUpdate(sql);

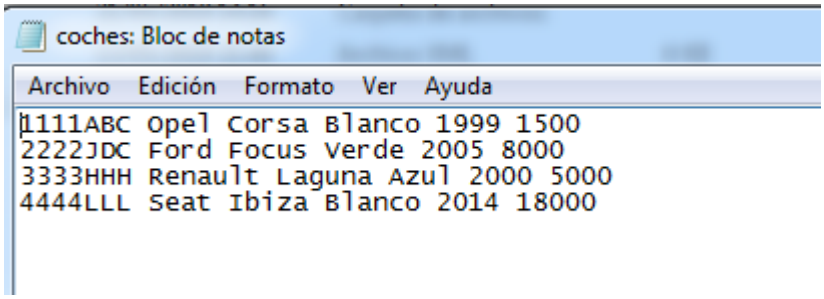
Esto crea la table coche (Siempre que no haya errores como los previstos en los catch)

```
43
44      public static void cargarCoches()
45      {
46          BufferedReader br = null;
47          String cadena;
48          try
49          {
50              br = new BufferedReader(new FileReader(new File("coches.txt")));
51
52              while ( (cadena = br.readLine()) != null)
53              {
54                  String[] campos = cadena.split(" ");
55                  Coche coche = new Coche(
56                          campos[0], campos[1], campos[2], campos[3],
57                          Integer.parseInt(campos[4]), Double.parseDouble(campos[5]));
58
59                  insertarCoche(coche);
60              }
61          } catch (IOException exc) {
62              System.err.println(exc.getMessage());
63          }finally
64          {
65              try
66              {
67                  if(br!=null)
68                     br.close();
69              }
70              catch(IOException ioe){
71                  System.err.println(ioe.getMessage());
72              }
73          }
74      }
75
```

El método cargarcoches() requiere que haya un archivo coches.txt desde el que leer para crear los objetos Coche y luego para guardarlos en la tabla mediante una llamada al método insertarCoche(). Primero se lee el documento y se separa mediante la orden Split por espacios, cada uno de los campos, después se usan esos campos para generar el objeto Coche, que se insertará mediante el método insertarCoche

Situación del archivo Coches.txt:






```
76      public static List<Coche> getCoches()
77      {
78          List<Coche> coches = new ArrayList();
79          Connection conexion = BaseDatos.getInstance().getConnection();
80          String sql = "select * from coches";
81
82          try
83          {
84              Statement sentencia = conexion.createStatement();
85              ResultSet rs = sentencia.executeQuery(sql);
86
87              while (rs.next())
88              {
89                  Coche c = new Coche(rs.getString(1), rs.getString(2),
90                          rs.getString(3), rs.getString(4), rs.getInt(5), rs.getDouble(6));
91                  coches.add(c);
92                  System.out.println(c);
93              }
94          }
95          catch (SQLException ex)
96          {
97              System.err.println("Error en el método getCoches");
98              System.err.println(ex.getMessage());
99          }
100         return coches;
101     }
102
```

El método getcoches() permite consultar toda la información que hay en la tabla de MySQL, creando una instancia de la clase Coche con cada una de las filas de la tabla, la cual se añadirá a una lista de objetos Coche llamada coches, que será la que el método devuelva. También hay que hacer notar que la línea System.out.println(c) nos escribirá la información del objeto

```
103     public static boolean existeCoche(String matricula)
104     {
105         Connection conexion = BaseDatos.getInstance().getConnection();
106         String sql = "select count(matricula) from coches where matricula = ?";
107
108         try
109         {
110             PreparedStatement ps = conexion.prepareStatement(sql);
111             ps.setString(1, matricula);
112             ResultSet rs = ps.executeQuery();
113             rs.next();
114             if(rs.getInt(1)>0)
115                 return true;
116         }
117         catch (SQLException ex)
118         {
119             System.err.println("Error en el método existeCoche");
120             System.err.println(ex.getMessage());
121         }
122
123         return false;
124     }
125
```

El método existeCoche() nos informa de si el coche cuya matrícula le pasamos existe o no. Se crea una conexión a la base de datos y se crea una cadena de caracteres en el que el lugar que correspondería a la matricula, le ponemos un

cierre de pregunta (interrogante). Después, mediante las siguientes tres líneas, preparamos la consulta y la ejecutamos:

PreparedStatement ps = conexion.prepareStatement(sql);

ps.setString(1, matricula);

ResultSet rs = ps.executeQuery();

```
127     private static void insertarCoche(Coche c)
128     {
129         if (!existeCoche(c.getMatricula()))
130         {
131             Connection conexion = BaseDatos.getInstance().getConnection();
132             String sql = "insert into coches values (?,?,?,?,?,?)";
133             try
134             {
135                 PreparedStatement ps = conexion.prepareStatement(sql);
136 
137                 ps.setString(1, c.getMatricula());
138                 ps.setString(2, c.getMarca());
139                 ps.setString(3, c.getModelo());
140                 ps.setString(4, c.getColor());
141                 ps.setInt(5, c.getAño());
142                 ps.setDouble(6, c.getPrecio());
143 
144                 ps.executeUpdate();
145             }
146             catch (SQLException exc)
147             {
148                 System.err.println(exc.getMessage());
149             }
150         }
151     }
152 
153 }
154 
```

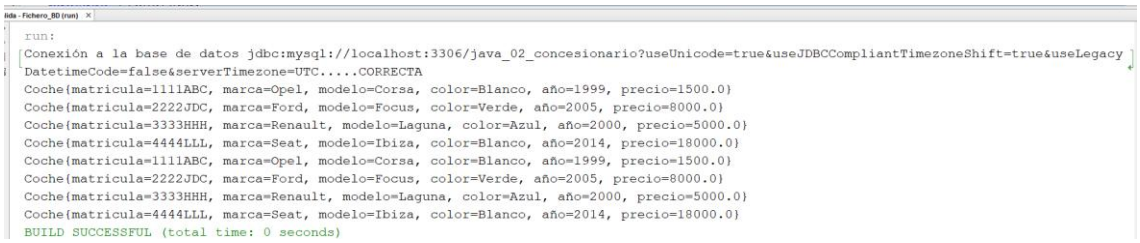
Este método permite insertar un Coche en la tabla, si el coche no existe en la tabla (comparándolo mediante c.getMatricula()) se crea una conexión y una cadena de caracteres que almacena una consulta, después como en el método anterior, preparamos y ejecutamos.

La clase principal

```java
package ficherobd;

/**
 *
 * @author
 */
public class Principal
{

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        MetodosBBDD.crearTablaCoches();
        MetodosBBDD.cargarCoches();
        for(Coche coche: MetodosBBDD.getCoches())
            System.out.println(coche);
    }

}
```

Se crea la tabla, se cargan los coches y se lee la lista que proporciona el método getCoches(), sacando la información de los mismos por consola.

La ejecución de este programa producirá el siguiente resultado:



Si eliminamos la línea de getCoches() en la que se escriben los coches, tenemos: