

ACCESO A DATOS - T1 - HITO 1

Sergio Camino Saiz

18/10/2022



ÍNDICE

TECNOLOGÍAS USADAS	2
SCRIPT SQL UTILIZADO	3
EJECUCIÓN DEL PROGRAMA	5
EXPLICACIÓN DEL CÓDIGO	6



TECNOLOGÍAS USADAS

Las tecnologías que he utilizado son las siguientes :

- Bases de datos : MySQL Workbench y PostgreSQL.
- IDE : IntelliJ.
- Lenguaje: Java.
- Conectores para las bases de datos: MySQL (dependencia dentro del archivo POM.xml) y PostgreSQL (archivo '.jar' introducido dentro de la estructura de proyectos en el apartado 'Modules/Dependencies').
- Archivos CSV para almacenar la información.

SCRIPT SQL UTILIZADO

Las bases de datos son las mismas en ambos DBMS. Los scripts de las tablas están adjuntados al proyecto.

MySQL :



Tabla t_coche :

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_coche	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
marca_coche	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
modelo_coche	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
matricula_coche	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Tabla t_propietario :

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_propietario	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
nombre_propietario	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
telefono_propietario	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
idcoche_propietario	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Foreign Key Name	Referenced Table
id_coche	`coches`, `t_coches`
<input type="text"/>	

PostgreSQL :

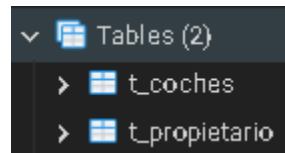
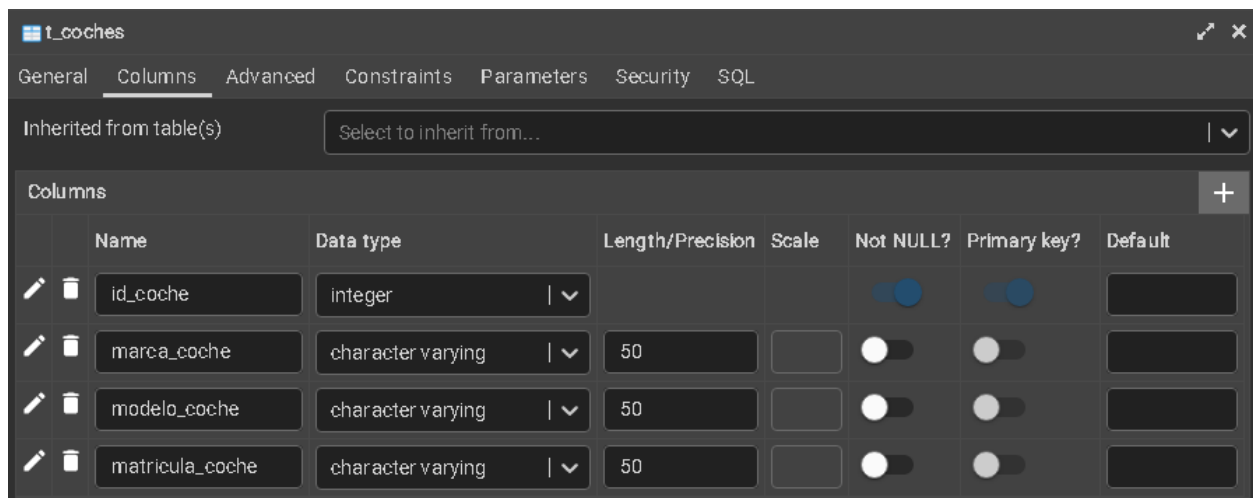


Tabla t_coches:











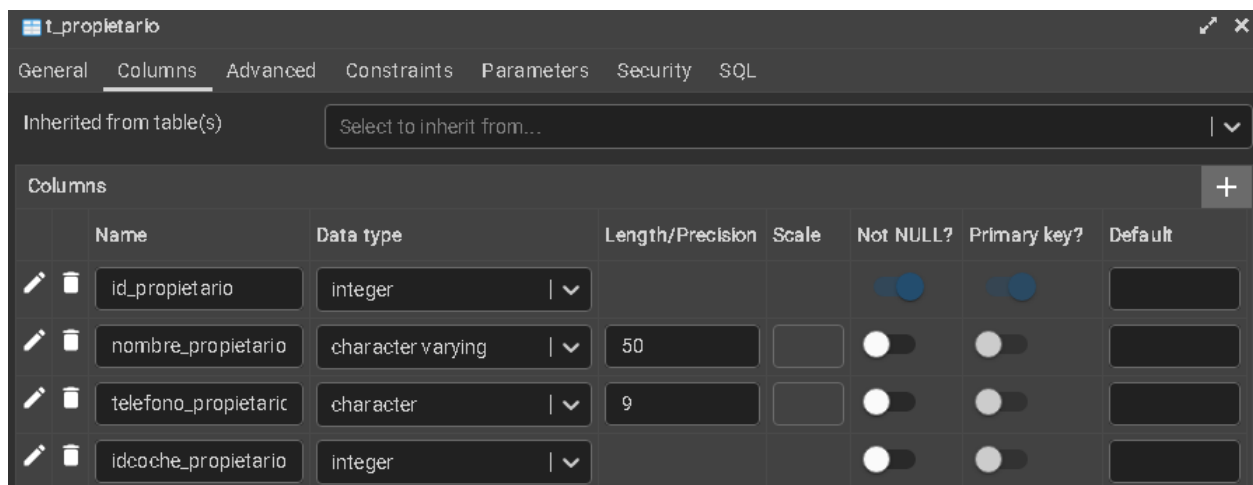








	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
 	id_coche	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
 	marca_coche	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	
 	modelo_coche	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	
 	matricula_coche	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	

Tabla t_propietario:



	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
 	id_propietario	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
 	nombre_propietario	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	
 	telefono_propietario	character	9		<input type="checkbox"/>	<input type="checkbox"/>	
 	idcoche_propietario	integer			<input type="checkbox"/>	<input type="checkbox"/>	

EJECUCIÓN DEL PROGRAMA

```
Se ha establecido la conexión con la base de datos.  
1 : Añadir coche  
2 : Mostrar coches  
3 : Actualizar coche  
4 : Eliminar coche  
5 : Añadir propietario  
6 : Mostrar propietarios  
7 : Actualizar propietario  
8 : Eliminar propietario  
9 : Importar los datos a CSV  
10 : Generar copia de seguridad de COCHES en CSV  
11 : Generar copia de seguridad de PROPIETARIOS en CSV  
12 : Importar datos a la tabla COCHES  
13 : Importar datos a la tabla PROPIETARIOS  
14 : Cerrar programa
```

```
📄 copiaSeguridad_MySQL_COCHES.csv  
📄 copiaSeguridad_MySQL_PROPIETARIO.csv  
📄 copiaSeguridad_Postgre_COCHES.csv  
📄 copiaSeguridad_Postgre_PROPIETARIO.csv  
📄 datos_MySQL.csv  
📄 datos_Postgre.csv
```

EXPLICACIÓN DEL CÓDIGO

Tengo 2 archivos 1 para cada base de datos.

MySQL :

Aquí realizo la conexión entre el programa y la base de datos. En ese bloque de código le indico el tipo de base de datos, el nombre de la base de datos, el conector que utilizare para realizar la conexión y el usuario y contraseña con la que entraré.

```
// -- CONEXION CON LA BASE DE DATOS
// 2 : COMPROBAMOS QUE EL CONNECTOR ES RECONOCIDO.
try {
    Class.forName( className: "com.mysql.cj.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");

// 3 : USAMOS JDBC CONNECTOR PARA ESTABLECER CONEXION CON LA BASE DE DATOS.
String conexionbd = "jdbc:mysql://localhost:3306/coches";
String user = "root";
String pass = "curso";
Connection con;
try {
    con = DriverManager.getConnection(conexionbd, user, pass);
} catch (SQLException e) {
    System.out.println("No se ha podido establecer la conexión con la base de datos.");
    System.out.println(e.getMessage());
    return;
}
System.out.println("Se ha establecido la conexión con la base de datos.");
```

Tengo un menú por consola para que el usuario escoja la opción que más desee.

```
// -- MENU POR CONSOLA
Scanner lector = new Scanner(System.in);
while(!lector.equals("14")) {
    System.out.println("1 : Añadir coche ");
    System.out.println("2 : Mostrar coches ");
    System.out.println("3 : Actualizar coche ");
    System.out.println("4 : Eliminar coche ");
    System.out.println("5 : Añadir propietario ");
    System.out.println("6 : Mostrar propietarios ");
    System.out.println("7 : Actualizar propietario ");
    System.out.println("8 : Eliminar propietario ");
    System.out.println("9 : Importar los datos a CSV");
    System.out.println("10 : Generar copia de seguridad de COCHES en CSV");
    System.out.println("11 : Generar copia de seguridad de PROPIETARIOS en CSV");
    System.out.println("12 : Importar datos a la tabla COCHES");
    System.out.println("13 : Importar datos a la tabla PROPIETARIOS");
    System.out.println("14 : Cerrar programa ");
}
```

Las opciones de CRUD (1-8) son sentencias SQL utilizando ‘Statement’, ‘ResultSet’, etc.

En el 'case 9' exporto los datos de ambas tablas a un archivo CSV. Utilizo clases como 'PrintWriter' para crear el archivo, StringBuilder para almacenar las cadenas de textos y 'ResultSet' para realizar la sentencia select a la tabla. Todo en ambos archivos '.java' (tanto en MySQL como en PostgreSQL).

```
//CREAMOS EL FICHERO DONDE GUARDAREMOS LOS DATOS.
PrintWriter pw= new PrintWriter(new File( pathname: "datos_MySQL.csv"));
//UTILIZAMOS UN OBJETO StringBuilder PARA ALMACENAR CADENAS DE TEXTO.
StringBuilder sb=new StringBuilder();

//SENTENCIA SELECT DONDE ALMACENARE LOS DATOS EN MEMORIA.
ResultSet rs2=null;
String query="select * from t_coches";
PreparedStatement ps=con.prepareStatement(query);
rs2=ps.executeQuery();

//MIENTRAS EL RESULTSET TENGA UN SIGUIENTE OBJETO CREA UN "HIJO" CON
while(rs2.next()){
    sb.append(rs2.getInt( columnIndex: 1));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 2));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 3));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 4));
    //HACEMOS UN SALTO DE LINEA POR CADA OBJETO QUE PINTEMOS.
    sb.append("\r\n");
}
System.out.println("Los coches han sido almacenados.");
```

Los case 10 y 11 lo que hacen es generar un archivo CSV donde guardaremos los datos de las tablas. Son 2 case porque utilizó un case por tabla. Los case son iguales tan solo se cambia los nombres de los archivos, el tipo de dato que almacena y las sentencias.

```
//COPIA DE SEGURIDAD.
//CREAMOS EL FICHERO DONDE GUARDAREMOS LOS DATOS.
PrintWriter pw= new PrintWriter(new File( pathname: "copiaSeguridad_Postgre_COCHES.csv"));
//UTILIZAMOS UN OBJETO StringBuilder PARA ALMACENAR CADENAS DE TEXTO.
StringBuilder sb=new StringBuilder();

//SENTENCIA SELECT DONDE ALMACENARE LOS DATOS EN MEMORIA.
ResultSet rs2=null;
String query="select * from t_coches";
PreparedStatement ps=con.prepareStatement(query);
rs2=ps.executeQuery();

//MIENTRAS EL RESULTSET TENGA UN SIGUIENTE OBJETO CREA UN "HIJO" CON STRINGBUILDER.
while(rs2.next()){
    sb.append(rs2.getInt( columnIndex: 1));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 2));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 3));
    sb.append(",");
    sb.append(rs2.getString( columnIndex: 4));
    //HACEMOS UN SALTO DE LINEA POR CADA OBJETO QUE PINTEMOS.
    sb.append("\r\n");
}
System.out.println("Los coches han sido almacenados.");
//GUARDAMOS LOS DATOS DEL STRINGBUILDER COMO UN STRING.
pw.write(sb.toString());
//CERRAMOS EL PRINTWRITER.
pw.close();
System.out.println("La copia de seguridad de Coches se ha generado con éxito");
```

```

//COPIA DE SEGURIDAD.
//CREAMOS EL FICHERO DONDE GUARDAREMOS LOS DATOS.
PrintWriter pw= new PrintWriter(new File( pathname: "copiaSeguridad_Postgre_PROPIETARIO.csv"));
//UTILIZAMOS UN OBJETO StringBuilder PARA ALMACENAR CADENAS DE TEXTO.
StringBuilder sb=new StringBuilder();

//SENTENCIA SELECT DONDE ALMACENARE LOS DATOS EN MEMORIA.
ResultSet rsP=null;
String queryP="select * from t_propietario";
PreparedStatement psP=con.prepareStatement(queryP);
rsP=psP.executeQuery();

//MIENTRAS EL RESULTSET TENGA UN SIGUIENTE OBJETO CREA UN "HIJO" CON STRINGBUILDER.
while(rsP.next()){
    sb.append(rsP.getInt( columnIndex: 1));
    sb.append(",");
    sb.append(rsP.getString( columnIndex: 2));
    sb.append(",");
    sb.append(rsP.getString( columnIndex: 3));
    sb.append(",");
    sb.append(rsP.getInt( columnIndex: 4));
    //HACEMOS UN SALTO DE LINEA POR CADA OBJETO QUE PINTEMOS.
    sb.append("\r\n");
}

System.out.println("Los propietarios han sido almacenados");
//GUARDAMOS LOS DATOS DEL STRINGBUILDER COMO UN STRING.
pw.write(sb.toString());
//CERRAMOS EL PRINTWRITER.
pw.close();
System.out.println("La copia de seguridad de Propietarios se realizo con éxito");

```

Los case 12 y 13 lo que hacen es importar los datos que contienen los archivos csv (las copias de seguridad) a las tablas de la base de datos. Realizó la conexión con la base de datos, específico el archivo que utilizaremos, realizó la sentencia 'Insert into' y le específico los datos que debe ingresar (con la conversión del tipo de dato) y finalmente ejecutamos la sentencia. La estructura es la misma para los 2 case tan solo se cambia los archivos csv, la sentencia 'Insert into' y los tipos de datos.

```
//COPIA DE SEGURIDAD.
//CREAMOS EL FICHERO DONDE GUARDAREMOS LOS DATOS.
PrintWriter pw= new PrintWriter(new File( pathname: "copiaSeguridad_Postgre_PROPIETARIO.csv"));
//UTILIZAMOS UN OBJETO StringBuilder PARA ALMACENAR CADENAS DE TEXTO.
StringBuilder sb=new StringBuilder();

//SENTENCIA SELECT DONDE ALMACENARE LOS DATOS EN MEMORIA.
ResultSet rsP=null;
String queryP="select * from t_propietario";
PreparedStatement psP=con.prepareStatement(queryP);
rsP=psP.executeQuery();

//MIENTRAS EL RESULTSET TENGA UN SIGUIENTE OBJETO CREA UN "HIJO" CON STRINGBUILDER.
while(rsP.next()){
    sb.append(rsP.getInt( columnIndex: 1));
    sb.append(",");
    sb.append(rsP.getString( columnIndex: 2));
    sb.append(",");
    sb.append(rsP.getString( columnIndex: 3));
    sb.append(",");
    sb.append(rsP.getInt( columnIndex: 4));
    //HACEMOS UN SALTO DE LINEA POR CADA OBJETO QUE PINTEMOS.
    sb.append("\r\n");
}

System.out.println("Los propietarios han sido almacenados");
//GUARDAMOS LOS DATOS DEL STRINGBUILDER COMO UN STRING.
pw.write(sb.toString());
//CERRAMOS EL PRINTWRITER.
pw.close();
System.out.println("La copia de seguridad de Propietarios se realizo con éxito");
```

```

try{
    con = DriverManager.getConnection(conexionbd, user, pass);
    con.setAutoCommit(false);

    String csvFilePath2 = "copiaSeguridad_Postgre_PROPIETARIO.csv";

    int batchSize = 20;

    String sql = "INSERT INTO t_propietario (id_propietario, nombre_propietario, telefono_pr
    PreparedStatement statement2 = con.prepareStatement(sql);

    BufferedReader lineReader2 = new BufferedReader(new FileReader(csvFilePath2));
    String lineText = null;

    int count = 0;

    lineReader2.readLine(); // skip header line

    while ((lineText = lineReader2.readLine()) != null) {
        String[] data = lineText.split( regex: ",");
        String id_propietario = data[0];
        String nombre_propietario = data[1];
        String telefono_propietario = data[2];
        String idcoche_propietario = data[3];

        int Intid_coche = Integer.parseInt(id_propietario);
        statement2.setInt( parameterIndex: 1, Intid_coche);
        statement2.setString( parameterIndex: 2, nombre_propietario);
        statement2.setString( parameterIndex: 3, telefono_propietario);
        int Intidcoche_propietario = Integer.parseInt(idcoche_propietario);
        statement2.setInt( parameterIndex: 4, Intidcoche_propietario);
    }
}

```

```

        statement2.addBatch();

        if (count % batchSize == 0) {
            statement2.executeBatch();
        }
    }

    lineReader2.close();

    // execute the remaining queries
    statement2.executeBatch();
    con.commit();
    System.out.println("Los propietarios han sido importados a la base de datos con éxito");
}

```

La clase 14 lo que hace es cerrar el programa. El programa después de elegir una opción no se cierra, por eso este case se utiliza para cerrar el programa.

```

//CERRAR LA CONEXION CON LA BASE DE DATOS Y SALIR DEL MENU
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");
break;

```

PostgreSQL :

Para la conexión entre el programa y PostgreSQL he utilizado lo siguiente. Primero me descargue el conector (.jar) y se lo añadí al proyecto en 'Modules/Dependencies'. Después realice la conexión mediante el siguiente código.

```
// -- CONEXION CON LA BASE DE DATOS
// COMPROBAMOS QUE EL CONNECTOR ES RECONOCIDO.
try {
    Class.forName( className: "org.postgresql.Driver");
} catch (ClassNotFoundException ex) {
    System.out.println("Error al registrar el driver de PostgreSQL: " + ex);
}

// 3 : USAMOS JDBC CONNECTOR PARA ESTABLECER CONEXION CON LA BASE DE DATOS.
String conexionbd = "jdbc:postgresql://localhost:5432/coches";
String user = "postgres";
String pass = "curso";
Connection con;
try {
    con = DriverManager.getConnection(conexionbd, user, pass);
} catch (SQLException e) {
    System.out.println("No se ha podido establecer la conexión con la base de datos.");
    System.out.println(e.getMessage());
    return;
}
System.out.println("Se ha establecido la conexión con la base de datos.");
```

El código que he utilizado en la clase Postgresql.java es el mismo que he explicado anteriormente, tan solo se cambia la conexión a la base de datos.