Argentina programa

Desarrollo de aplicaciones JAVA

Guía IV

"Acceso a Datos con JDBC"

PRÓLOGO: DESCUBRIENDO EL PODER DEL TRABAJO EN EQUIPO

Estimados alumnos,

Es un placer darles la bienvenida a esta guía de estudios que marcará un nuevo capítulo en nuestro viaje de aprendizaje. A partir de este momento, nos embarcaremos en un emocionante camino donde el trabajo en equipo será la piedra angular de nuestro progreso.

En esta guía, nos adentraremos en el apasionante mundo del desarrollo de proyectos, una experiencia en la que cada uno de ustedes desempeñará un papel fundamental. Nuestro objetivo será trascender las fronteras del aprendizaje individual y unir nuestras habilidades y conocimientos para alcanzar logros asombrosos.

La guía propone el desarrollo de un proyecto ejemplo que nos servirá como punto de partida para explorar conceptos fundamentales en el diseño de bases de datos y la programación en Java. Sin embargo, lo más emocionante es que, como equipo, tendrán la oportunidad de llevar a cabo un proyecto similar, en el cual podrán poner en práctica todo lo aprendido y enfrentar desafíos reales.

A medida que avancemos, descubriremos la importancia de la colaboración, la comunicación efectiva y el respeto mutuo en el trabajo en equipo. Aprenderemos a aprovechar las fortalezas individuales de cada miembro para construir un todo más poderoso y resiliente.

Recuerden que este viaje será tanto desafiante como gratificante. Habrá momentos en los que enfrentaremos obstáculos, pero estoy seguro de que juntos superaremos cada adversidad y alcanzaremos resultados que nos llenarán de orgullo.

Los invito a sumergirse en esta guía de estudios con mente abierta y corazón dispuesto a colaborar. Estoy entusiasmado de ver cómo se desarrolla nuestro proyecto conjunto y los talentos que cada uno de ustedes aportará a él. ¡Juntos alcanzaremos nuevas alturas y construiremos un legado duradero!

INTRODUCCIÓN

JDBC (Java DataBase Connectivity) es la tecnología Java que permite a las aplicaciones interactuar directamente con motores de base de datos relacionales.

La API JDBC es una parte integral de la plataforma Java, por lo tanto no es necesario descargar ningún paquete adicional para usarla. JDBC es una interface única que independiza a las aplicaciones del motor de la base de datos.

Un driver JDBC es usado por la JVM (Java Virtual Machine) para introducir las invocaciones JDBC en invocaciones que la base de datos entiende.

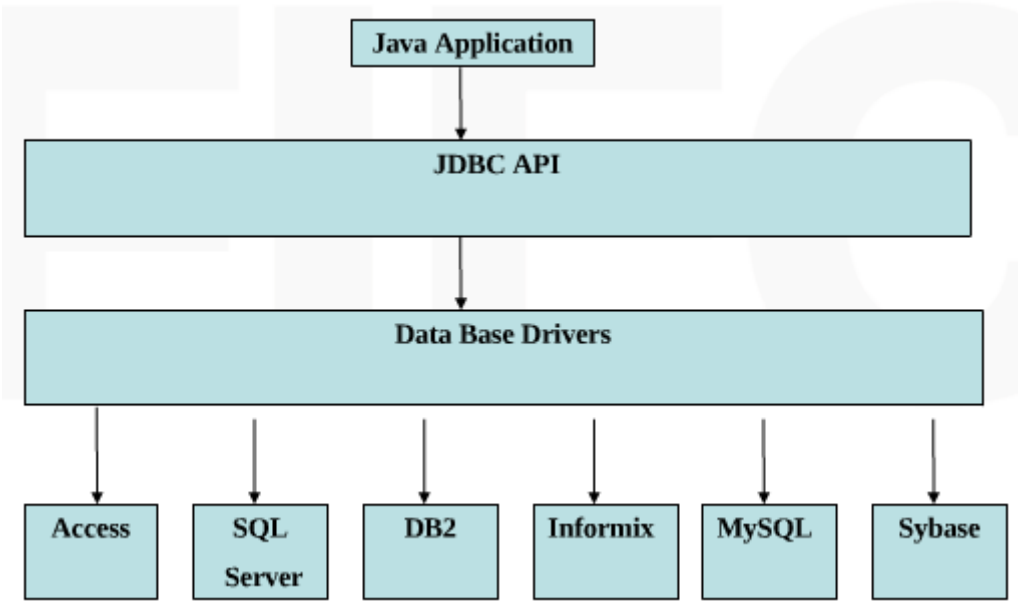


Ilustración 1-Drivers de conexión

Como se muestra en la "Ilustración 1", agregaremos a nuestro proyecto un Driver de Conexión de acuerdo al motor de bases de datos que vaya a utilizar nuestra aplicación; es decir, si nuestra aplicación se tiene que conectar a una base de datos MariaDB, agregaremos a nuestro proyecto, un driver de conexión para bases de datos MariaDB; el cual descargaremos, generalmente, desde el sitio oficial de MariaDB; este es un tema que veremos más adelante en detalle.

¿Qué es JDBC?

Consta de un conjunto de clases e interfaces Java que nos permiten acceder de una forma genérica a las bases de datos independientemente del proveedor del SGBD (Sistema Gestor de Base de Datos).

Cada proveedor dispondrá de una implementación de dichas interfaces. Dicha implementación, se sabe comunicar con el SGBD de ese proveedor. (*Ilustración 1).*

Se encuentra en el paquete java.sql.*

Básicamente una aplicación que utiliza JDBC realiza los siguientes pasos:

a. Establece una conexión con una base de datos.
b. Crea y envía una sentencia SQL a la base de datos.
c. Procesa el resultado.

¿Qué es una aplicación de dos capas?

Una aplicación de dos capas es una en la cual el código está separado del sistema de administración de la base de datos. Esas capas pueden estar separadas lógicamente, es decir cada una ejecuta en un proceso diferente en la misma computadora, o separadas físicamente, es decir cada una ejecuta en procesos diferentes en computadoras diferentes.

Una aplicación de dos capas, realmente tiene tres partes: un cliente, un servidor y un protocolo. El protocolo es el puente entre las capas del cliente y el servidor. En la capa del cliente reside la interfaz del usuario y la lógica del negocio, mientras que en el servidor está un administrador de bases de datos.

El protocolo en una aplicación de dos capas comunica una capa con la otra, que en ocasiones pueden residir en dos computadoras diferentes. El cliente le hace solicitudes al servidor y el servidor le regresa al cliente los resultados de esas solicitudes.

En Java, ese protocolo toma la forma de una API para facilitarnos la construcción de aplicaciones cliente – servidor. Los métodos de las clases de esta API establecen un mecanismo estándar para que un cliente escrito en Java se comunique con administradores de bases de datos de diferentes fabricantes, como habíamos mencionado anteriormente.

En la Ilustración 2 se muestra un diagrama de clases parcial de las interfaces y clases que forman parte de la API java:

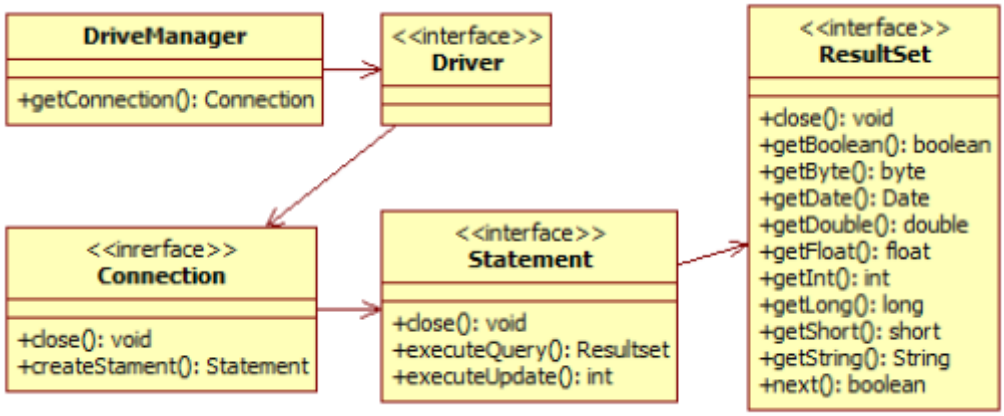


Ilustración 2Diagrama parcial API JDBC

• java.sql.Driver: Esta interfaz declara los métodos que toda clase que implementa un manejador JDBC debe tener. Normalmente los manejadores son

proporcionados por los fabricantes de administradores de bases de datos. En nuestro caso utilizaremos el driver de conexión MariaDB [1]como habíamos dicho.

• java.sql.DriverManager: Es la clase que permite la administración de los manejadores JDBC. DriveManager cargará a memoria el manejador JDBC de cada base de datos a la que nuestro programa desea comunicarse. Para cargar un manejador JDBC a memoria, DriveManager crea una instancia de la clase que implementa al manejador. Sin embargo en lugar de utilizar el operador *new* para crear la instancia de la clase, DriveManager utiliza un mecanismo que permite especificar el nombre de la clase en tiempo de ejecución en lugar de en tiempo de compilación. Este mecanismo permite crear una instancia de una clase dado el nombre de la clase y utiliza la siguiente sintaxis:

Class.forName("*nombreManejadorJDBC*").newInstance();

Donde *nombreManejadorJDBC* es el nombre del manejador JDBC.

Una vez que se ha cargado el manejador JDBC en memoria necesitamos establecer una conexión entre la API JDBC y el administrador de la base de datos. Para ello invocamos al método estático getConnection() de DriveManager.

• java.sql.Connection: Esta interfaz declara los métodos que debe tener una clase que represente una conexión lógica entre la API JDBC y la base de datos. Esta conexión nos permiten enviarle al manejador de la base de datos sentencias SQL y recuperar las repuestas provenientes desde el gestor de bases de datos.

Estas sentencias se encapsulan en objetos que implementan la interfaz java.sql.Statement. Para crear una sentencia SQL Connection tiene un método llamado prepareStatement.

• java.sql.Statement: Esta interfaz declara los métodos que debe tener una clase que represente el mecanismo que nos permita enviarle al manejador de la base de datos sentencias SQL y recuperar las repuestas del gestor de bases de datos.

• java.sql.ResultSet: Esta interfaz declara los métodos que debe tener una clase que represente una tabla de datos producida por una consulta a la base de datos. Un objeto de esta clase mantiene un cursor apuntando al renglón actual de la tabla. Inicialmente, el cursor está posicionado antes del primer renglón. Para obtener el siguiente renglón de la tabla se invoca al método next(). La

[1] **MariaDB** es una alternativa a MySQL **de** código abierto y gratuito.

interfaz declara métodos que permiten recuperar el valor de una columna de un renglón de la tabla.

Sintetizando, podemos decir que los **componentes de JDBC** son:

- El gestor de los drivers (java.sql.DriverManager).
- La conexión con la base de datos (java.sql.Connection).
- La sentencia a ejecutar (java.sql.Statement).
- El resultado (java.sql.ResultSet).

Para obtener una comprensión más clara de cómo se utilizan los componentes de JDBC, exploremos un ejemplo simple:

Sistema de gestión para la Universidad de La Punta:

La Universidad de La Punta cree necesario utilizar un sistema para poder llevar el registro de los alumnos de la institución y las materias que se dictan en la misma. Adicionalmente se necesita poder registrar las materias que cursa cada alumno. El sistema debe permitir cargar la calificación obtenida (nota) cuando un alumno rinde un examen final. Para cada materia que cursa un alumno solo se registrará la última calificación obtenida, o sea no se mantiene registro de las notas obtenidas anteriormente, por lo que, si un alumno rinde el examen final de una materia y obtiene una calificación de "2", y luego rinde nuevamente el examen para la materia y obtiene una calificación de "9" solo quedará registro de esta última.

Funcionalidad: el sistema deberá

1. Permitir al personal administrativo listar las materias que cursa un alumno.
2. Permitir al personal administrativo listar los alumnos inscriptos en una determinada materia.
3. Permitir que un alumno se pueda inscribir o des-inscribir en las materias que desee.
4. Permitir registrar la calificación final de una materia que está cursando un alumno.
5. Permitir el alta, baja y modificación de los alumnos y las materias.

Modelo de BD sugerido.

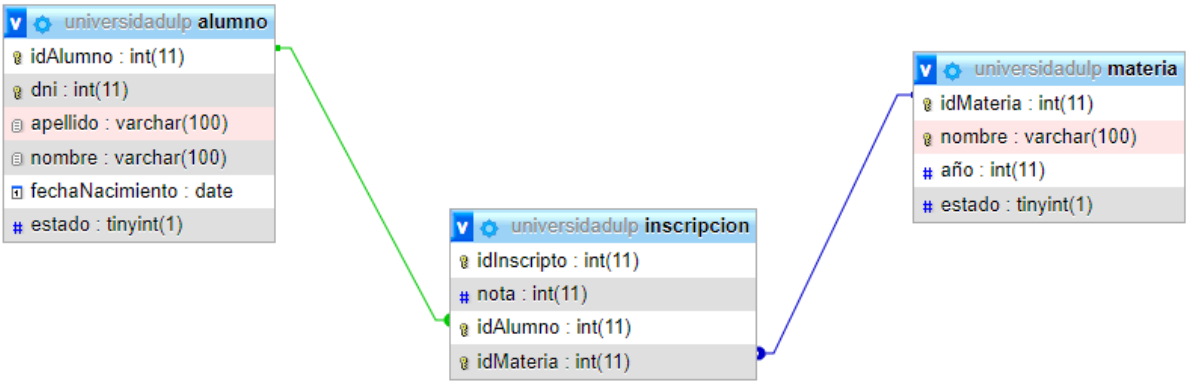


Ilustración 3Modelo Relacional Universidad

En este modelo podemos observar que podemos en la base de datos almacenar los datos de los alumnos, en donde su clave primaria es idAlumno un campo auto incremental, dni como valor único, apellido, nombre, fecha de nacimiento y un campo booleano, estado, que utilizaremos para las bajas lógicas, es decir, este campo tomará valor "false" cuando el alumno esté de baja y un valor "true" mientras esté activo; además podremos almacenar de las materias, su clave que es idMateria, también auto incremental, el nombre de la materia como único, año en que se dicta y un estado, para indicar si está activa o dada de baja, igual que con los alumnos. En el caso de querer registrar una inscripción de un alumno en una materia determinada, lo podremos indicar en la tabla inscripción, que también posee una clave que es idInscripto, auto incremental, el id del alumno y de la materia en la que está inscripto, además de la última nota registrada.

Antes de comenzar a desarrollar esta actividad, deberías ver el video "**creación de BD de la universidad**", luego uno de los miembros del equipo con la ayuda de todos construye el esquema en MySql y luego lo exporta a los demás miembros, para que todos compartan lo mismo; obviamente, si sucede que una vez finalizado hay que hacer algún tipo de cambio sobre el esquema de la base de datos, lo modifica uno de los miembros y nuevamente exporta el scrpit para el resto de sus compañeros de equipo.

Una vez que hayamos construido el esquema de la base de datos en MySQL, procederemos a utilizar JDBC para establecer la conexión y realizar diversas acciones a través de esta API en la base de datos. Para una mejor comprensión, llevaremos a cabo estos pasos directamente dentro del método main de un nuevo proyecto, el cual se explicará a continuación:

a) Crear en Netbeans un nuevo proyecto de nombre “Universidad Ejemplo”

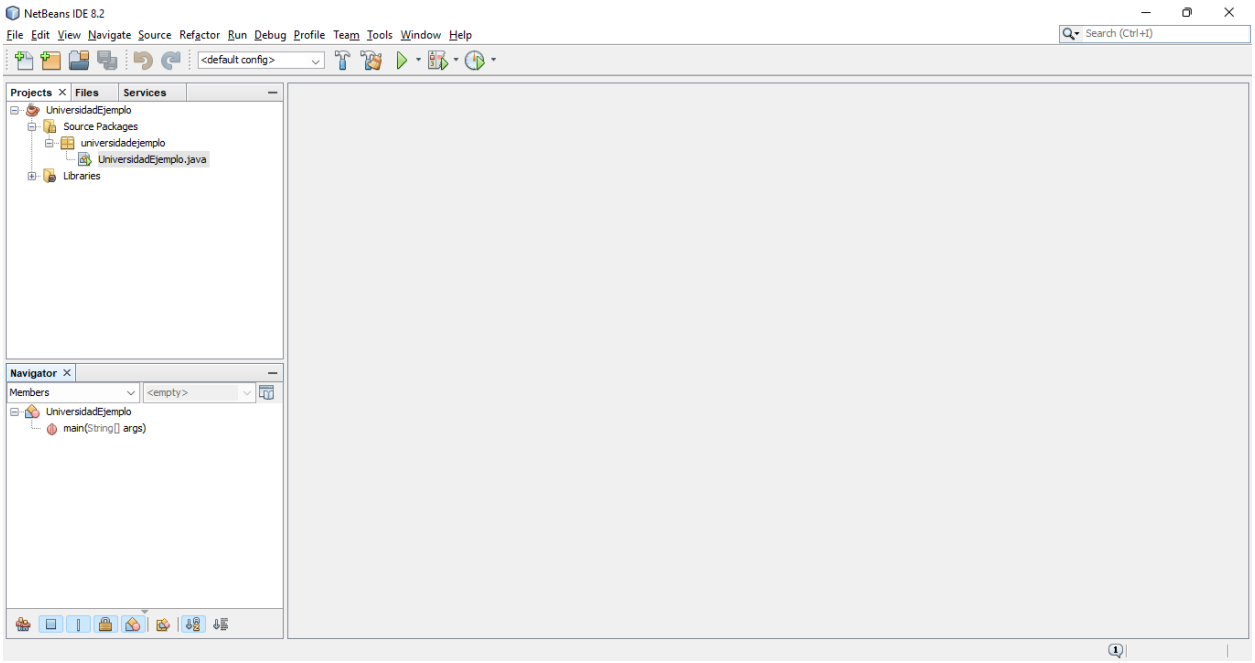


Ilustración 4 Proyecto nuevo

b) Descargamos el driver de conexión a la base de datos [MariaDB](https://mariadb.com/downloads/connectors/connectors-data-access/java8-connector) y lo agregamos al proyecto.

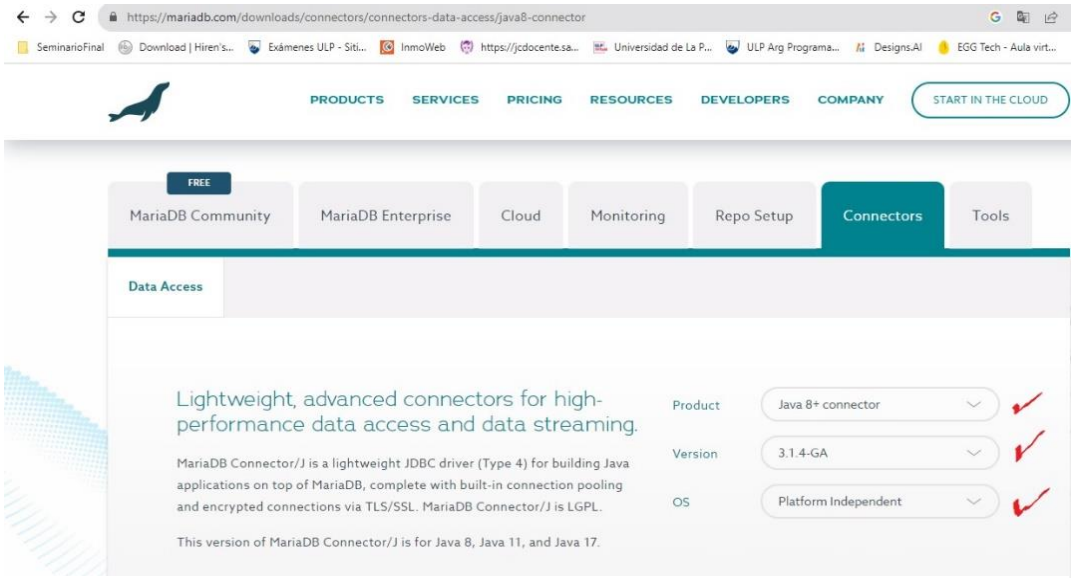


Ilustración 5 Web MariaDB

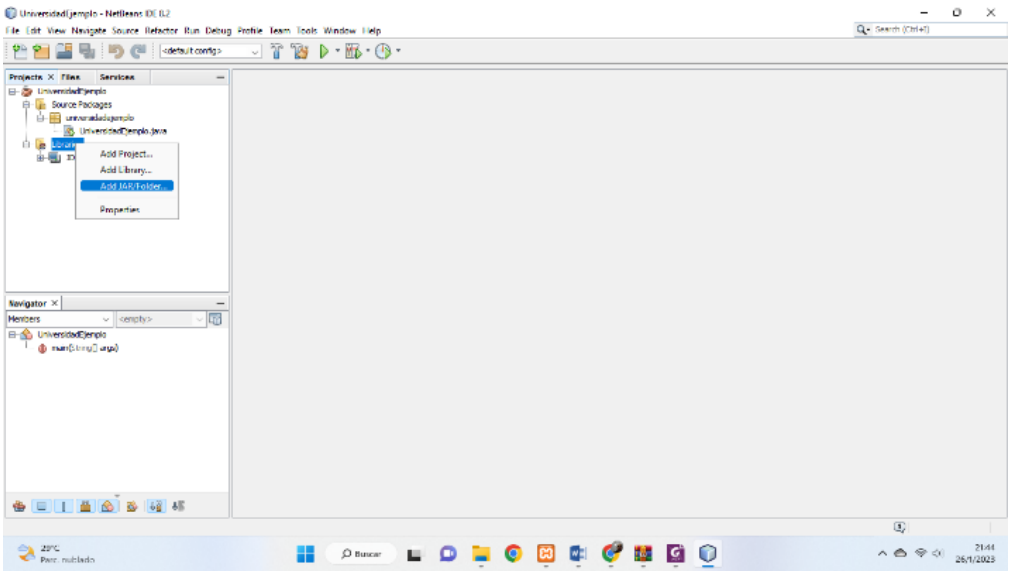


Ilustración 6 Agregando librería al proyecto

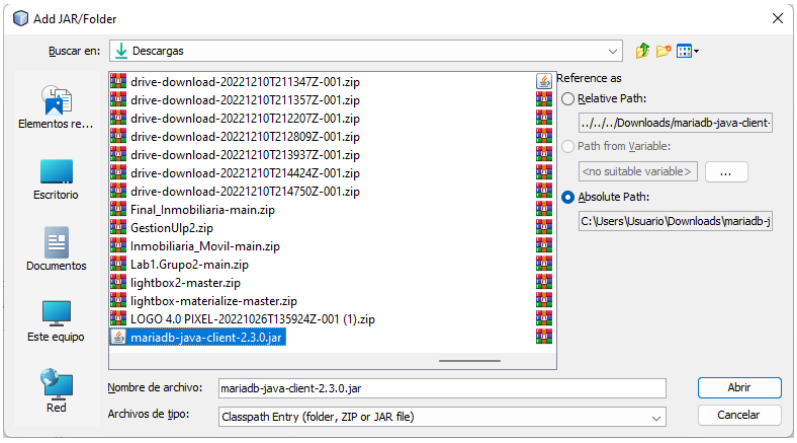


Ilustración 7 Agregando librería al proyecto II

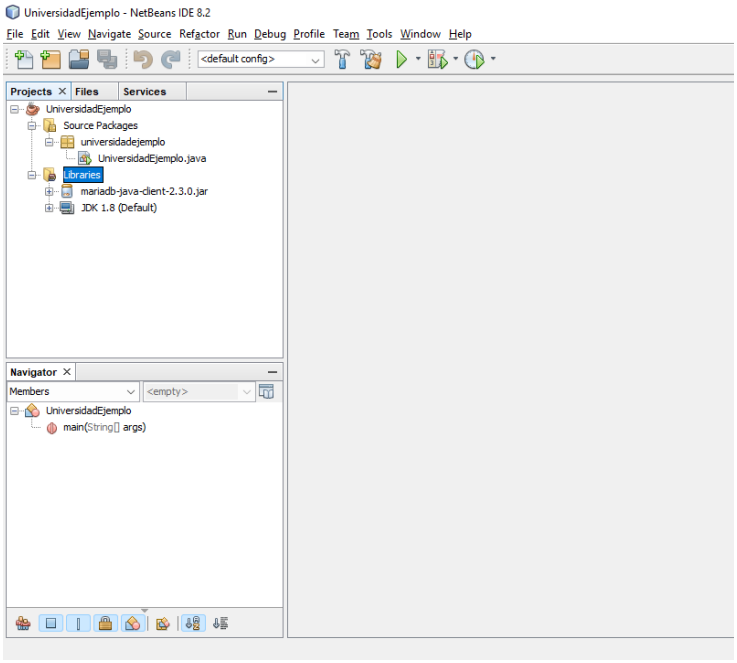


Ilustración 8 Librería agregada al proyecto

c) En el método main de este proyecto cargaremos el driver de conexión.

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
1
2  package universidadejemplo;
3
4
5  import javax.swing.*;
6
7
8  public class UniversidadEjemplo {
9
10
11      public static void main(String[] args) {
12          try {
13              //Cargar driver de conexión.
14              Class.forName("org.mariadb.jdbc.Driver");
15          } catch (ClassNotFoundException ex) {
16              JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
17          }
18      }
19
20 }
```

Ilustración 9

En la línea 14, el método forName() puede producir la ClassNotFoundException si no han sido agregado al proyecto el driver de conexión, por eso, esa línea está encerrada en un bloque try-catch, para informar a través de un diálogo, en el caso de producirse la excepción.

d) En este curso, utilizaremos Xampp que es un servidor independiente de código libre que incluye un gestor de base de datos MySql y otras herramientas que aprenderás a utilizar más adelante; y es allí en donde tenemos nuestra base de datos, que en este ejemplo le pusimos de nombre: "*universidadulp*".

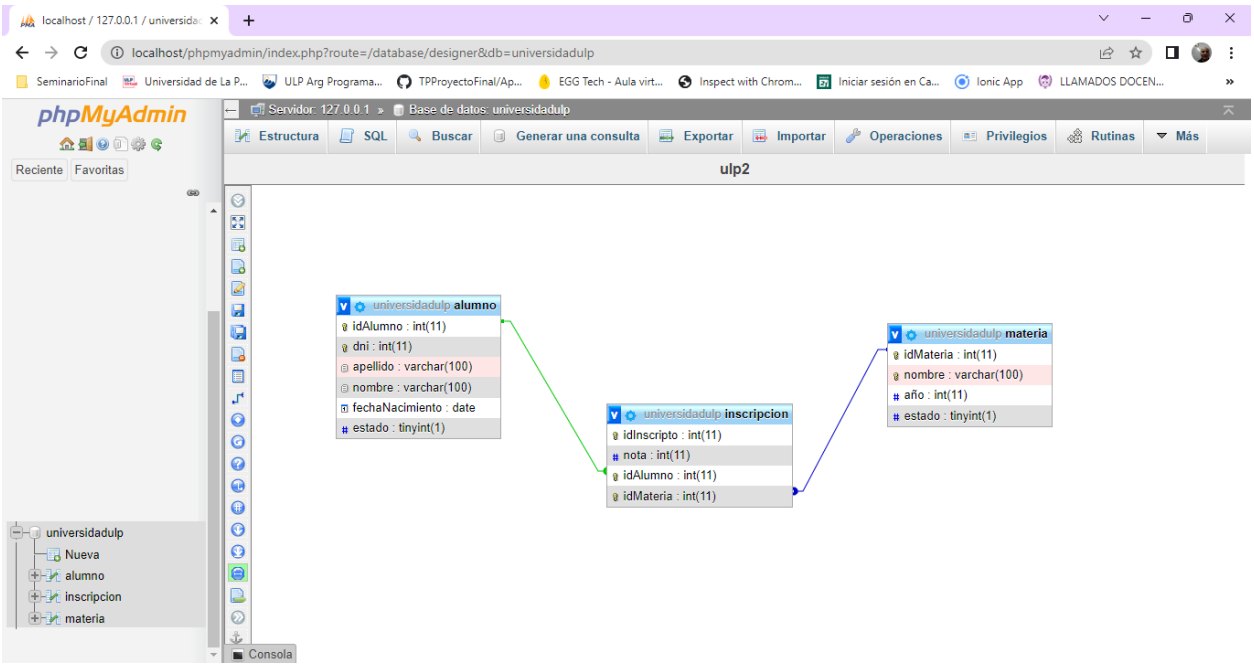


Ilustración 10

e) Ahora que nos queda claro que nuestra base de datos se llama “universidadulp” y el servidor está en nuestro hostlocal, vamos a establecer la conexión a la base de datos desde Java.

Para establecer la conexión, invocaremos al método estático getConnection() de la clase DriverManager con los siguientes parámetros:

DriverManager.getConnection("*jdbc:mysql://localhost/universidadulp*","*root*", "");
El String del primer parámetro “*jdbc:mysql://localhost/universidadulp”* indica que nos conectaremos a un gestor MySql que se encuentra en el hostLocal, en el caso de encontrarse el servidor en otra PC de la red, colocaremos allí el IP o nombre del host, luego seguido el nombre de la base de datos, en nuestro caso *universidadulp*. El String del segundo parámetro "*root*", corresponde al usuario por defecto del gestor de base de datos y el último String vacío es la contraseña por defecto.
Para terminar, la invocación al método getConnection(), nos devuelve un objeto de tipo Connection, que luego utilizaremos como mencionamos en los primeros párrafos, para hacer todas las operaciones sobre nuestra base de datos, como: consultas, inserciones, borrados y/o modificaciones. Además, este método, puede lanzar la SQLException en el caso de que la URL de conexión o el usuario y/o contraseña sean incorrectos.

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
 1
 2 package universidadejemplo;
 3
 4
 5 import java.sql.Connection;
 6 import java.sql.DriverManager;
 7 import java.sql.SQLException;
 8
 9 import javax.swing.*;
10
11
12 public class UniversidadEjemplo {
13
14
15  public static void main(String[] args) {
16     try {
17          //Cargar driver de conexión.
18          Class.forName("org.mariadb.jdbc.Driver");
19
20          //Conexión a la base de datos.
21         Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/universidadulp","root", "");
22
23      } catch (ClassNotFoundException ex) {
24             JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
25      } catch (SQLException ex) {
26             JOptionPane.showMessageDialog(null,"Error de Conexión");
27      }
28     }
29
30 }
```

Puede sustituir **mysql** por **mariadb**

Ilustración 11

f) Ahora agregaremos a nuestra base de datos una materia.

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
1
2 package universidadejemplo;
3
4
5 import java.sql.Connection;
6 import java.sql.DriverManager;
7 import java.sql.PreparedStatement;
8 import java.sql.SQLException;
9
10 import javax.swing.*;
11
12
13 public class UniversidadEjemplo {
14
15
16  public static void main(String[] args) {
17   try {
18        //Cargar driver de conexión.
19        Class.forName("org.mariadb.jdbc.Driver");
20
21        //Conexión a la base de datos.
22       Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/universidadulp","root", "");
23
24       String sql="insert into materia (nombre, año, estado) values ('Laboratorio 2', 2, true)";
25
26       PreparedStatement ps=conn.prepareStatement(sql);
27       int filas=ps.executeUpdate();
28       if(filas > 0){
29
30           JOptionPane.showMessageDialog(null,"Materia agregada Exitosamente");
31       }
32
33
34
35
36      } catch (ClassNotFoundException ex) {
37            JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
38      } catch (SQLException ex) {
39            JOptionPane.showMessageDialog(null,"Error "+ex.getMessage() );
40      }
41     }
42
43 }
```

Ilustración 12

Como se ve en la imagen, en la línea 24, se encuentra un String con la sentencias SQL que se enviará a la base de datos encargada de agregar una materia.

insert into materia (nombre, año, estado) values ('Laboratorio 2', 2, true)

Observe que no enviamos en la sentencia SQL el id de la materia, ya que este se genera automáticamente cuando se inserte la fila.

Luego en la línea 26, utilizando el objeto Connection que está siendo referenciado a través de la variable “conn” invocamos el método *prepareStatement()* y le pasamos como parámetro el String que contiene la sentencia SQL; este método generará un objeto de tipo

PreparedSatement, del cual invocaremos el método *executeUpdate()* que retorna un entero con la cantidad de filas afectadas; aunque este método también puede lanzar una SQLException si por ejemplo la sentencia sql enviada contiene algún error o infringe alguna restricción de la base de datos.

g) Agregamos un alumno

En este caso, en la línea 24 el String que forma la sentencia SQL con la orden de inserción concatena el dni, el apellido, el nombre cada uno de ellos entre comillas simples ‘’, además la fecha de nacimiento instanciada como un LocalDate y el valor booleano true que corresponde al estado de alumno activo.

```
String sql="insert into alumno (dni, apellido, nombre, fechaNacimiento, estado)"
        +"values(52745628,'Lopez','Juan', ' " + LocalDate.of(2000,Month.AUGUST,29)
+"',true);";
```

Como debería ser la sentencia SQL para insertar una inscripción??

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
 1
 2 package universidadejemplo;
 3
 4 import java.sql.Connection;
 5 import java.sql.DriverManager;
 6 import java.sql.PreparedStatement;
 7 import java.sql.SQLException;
 8 import java.time.LocalDate;
 9 import java.time.Month;
10 import javax.swing.*;
11
12 public class UniversidadEjemplo {
13
14  public static void main(String[] args) {
15   try {
16        //Cargar driver de conexión.
17        Class.forName("org.mariadb.jdbc.Driver");
18
19        //Conexión a la base de datos.
20       Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/universidadulp","root", "");
21
22       String sql="insert into alumno (dni, apellido, nombre, fechaNacimiento, estado)"
23               +"values (52745628, 'Lopez','Juan', '"+LocalDate.of(2000,Month.AUGUST,29)+"',true);";
24
25
26       PreparedStatement ps=conn.prepareStatement(sql);
27       int filas=ps.executeUpdate();
28       if(filas > 0){
29
30           JOptionPane.showMessageDialog(null,"Alumno agregado Exitosamente");
31       }
32
33
34
35
36      } catch (ClassNotFoundException ex) {
37            JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
38      } catch (SQLException ex) {
39            JOptionPane.showMessageDialog(null,"Error "+ex.getMessage() );
40      }
41     }
42
43 }
```

Ilustración 13

h) Obtener todos los alumnos activos y listarlos por consola.

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
 1
 2 package universidadejemplo;
 3
 4 import java.sql.Connection;
 5 import java.sql.DriverManager;
 6 import java.sql.PreparedStatement;
 7 import java.sql.ResultSet;
 8 import java.sql.SQLException;
 9 import java.time.LocalDate;
10 import java.time.Month;
11 import javax.swing.*;
12
13 public class UniversidadEjemplo {
14
15  public static void main(String[] args) {
16   try {
17         //Cargar driver de conexión.
18         Class.forName("org.mariadb.jdbc.Driver");
19
20         //Conexión a la base de datos.
21        Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/universidadulp","root", "");
22
23        String sql="select * from alumno where estado = true";
24
25
26        PreparedStatement ps=conn.prepareStatement(sql);
27        ResultSet resultado=ps.executeQuery();
28
29        while(resultado.next()){
30
31            System.out.println("Id "+resultado.getInt("idAlumno"));
32            System.out.println("DNI "+resultado.getInt("dni"));
33            System.out.println("Apellido "+resultado.getString("apellido"));
34            System.out.println("Nombre "+resultado.getString("nombre"));
35            System.out.println("Fecha de nacimiento "+resultado.getDate("fechaNacimiento"));
36            System.out.println("--------------------------------------------------------------------");
37        }
38
39
40
41       } catch (ClassNotFoundException ex) {
42             JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
43       } catch (SQLException ex) {
44             JOptionPane.showMessageDialog(null,"Error "+ex.getMessage() );
45       }
46      }
47
48 }
```

Ilustración 14

El String con la sentencia SQL para obtener todos los alumnos activos, es la siguiente:

*String sql="select * from alumno where estado = true";*

Si ejecutamos esta instrucción en el gestor de base de datos, obtendríamos algo como esto:

idAlumno	dni	apellido	nombre	fechaNacimiento	estado
3	333	Rodriguez	juan	2000-06-05	1
4	12345	Di Gangi	Yamila Belén	1996-01-06	1
5	52745628	Lopez	Juan	2000-08-29	1

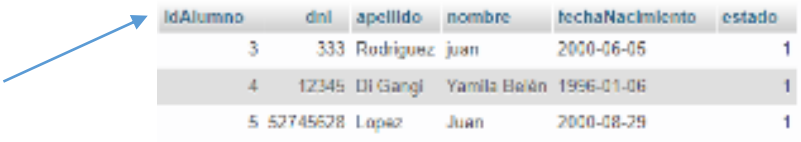
Ilustración 15 Ejemplar tabla alumno

En este ejemplar, las columnas son idAlumno, dni, apellido, nombre, fechaNacimiento y estado.

En la línea 26, invocamos el método prepareStatement a través del objeto Connection como lo veníamos haciendo normalmente, generando un objeto PreparedStatement, luego en la línea 27, a través de ese objeto PreparedStatement en lugar de invocar el método executeUpdate, invocamos el método excecuteQuery, porque la sentencia que estamos enviando a la base de datos es un **select**. El método excecuteQuery, puede lanzar la SQLException en el caso de que se encuentre mal armada la sentencia, por ejemplo mencionando una tabla que no existe o un campo que no existe; si todo está bien confeccionado, nos devuelve un ResultSet, que básicamente lo que contiene es lo mismo que nos devuelve el select en el gestor de base de datos, como lo muestra la ilustración 15. Este objeto ResultSet, dispone de varios métodos a través de los cuales, entre otras cosas, podemos recorrer cada fila y obtener un dato que se encuentra en una columna determinada de la fila sobre la que se está posicionado.

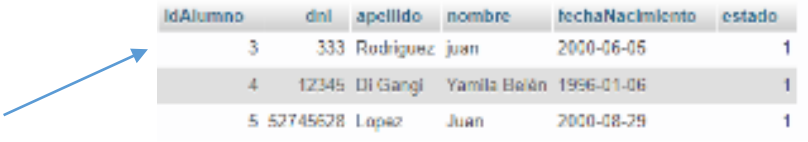
next():

El ResultSet posee un puntero posicionado fuera de la lista. El método **next()** devuelve true si hay una fila en donde pararse, y se para en esa fila.



idAlumno	dni	apellido	nombre	fechaNacimiento	estado
3	333	Rodriguez	juan	2000-06-05	1
4	12345	Di Gangi	Yamila Belén	1996-01-06	1
5	52745628	Lopez	Juan	2000-08-29	1

Es decir, si invocamos el método next() de este ResultSet, nos volverá true y se parará en la primer fila.



idAlumno	dni	apellido	nombre	fechaNacimiento	estado
3	333	Rodriguez	juan	2000-06-05	1
4	12345	Di Gangi	Yamila Belén	1996-01-06	1
5	52745628	Lopez	Juan	2000-08-29	1

Esta fila tiene la tupla (3, 333, Rodriguez, Juan, 2020-06-05,1), en la base de datos a los valores booleanos los representa con 0 (false) y 1(true).

getXXX():

El ResultSet, dispone de un método get por cada tipo de dato que deseamos extraer de una columna determinada, es decir, si deseamos extraer un entero utilizaremos el método getInt, si lo que deseamos extraer es el valor String de un determinado campo utilizaremos el método getString y así para cada tipo; pero, lo interesante de esto, es que estos métodos están sobrecargados, es decir, si quisiéramos extraer el id del alumno, que es un valor entero, podemos utilizar el método getInt pasándole como parámetro en forma de String el nombre del campo cuyo dato vamos a extraer o el número de columna en donde se encuentra, 1 para el id, 2 para el dni, 3 para el apellido, y así sucesivamente.

resultSet.getInt("dni");

ó

resultSet.getInt(1);

i) Dar de baja al alumno insertado en el punto anterior con dni: 52745628

Recordemos que la baja del alumno será una baja lógica, es decir, cambiarle el estado de true a false, no un borrado físico (delete). Por lo tanto, el String con la sentencia SQL que enviaremos a la base de datos sería la siguiente:

String sql="update alumno set estado = false where dni = 52745628";

C:/Users/Usuario/Documents/NetBeansProjects/UniversidadEjemplo/src/universidadejemplo/UniversidadEjemplo.java

```
 1
 2 package universidadejemplo;
 3
 4 import java.sql.Connection;
 5 import java.sql.DriverManager;
 6 import java.sql.PreparedStatement;
 7 import java.sql.ResultSet;
 8 import java.sql.SQLException;
 9 import java.time.LocalDate;
10 import java.time.Month;
11 import javax.swing.*;
12
13 public class UniversidadEjemplo {
14
15  public static void main(String[] args) {
16   try {
17        //Cargar driver de conexión.
18        Class.forName("org.mariadb.jdbc.Driver");
19
20        //Conexión a la base de datos.
21       Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/universidadulp","root", "");
22
23       String sql="update alumno set estado = false where dni = 52745628";
24
25
26       PreparedStatement ps=conn.prepareStatement(sql);
27       int fila=ps.executeUpdate();
28
29      if(fila>0){
30
31          JOptionPane.showMessageDialog(null,"Alumno dado de baja exitosamente!!!");
32      }
33
34
35
36      } catch (ClassNotFoundException ex) {
37            JOptionPane.showMessageDialog(null,"Debe agregar los driver al proyecto!!!");
38      } catch (SQLException ex) {
39            JOptionPane.showMessageDialog(null,"Error "+ex.getMessage() );
40      }
41     }
42
43 }
```

Nota: Cuando las sentencia que envío a la base de datos son: Insert. Update, Delete invoco el método executeUpdate; pero si envío un Select, utilizo executeQuery para obtener el ResultSet.