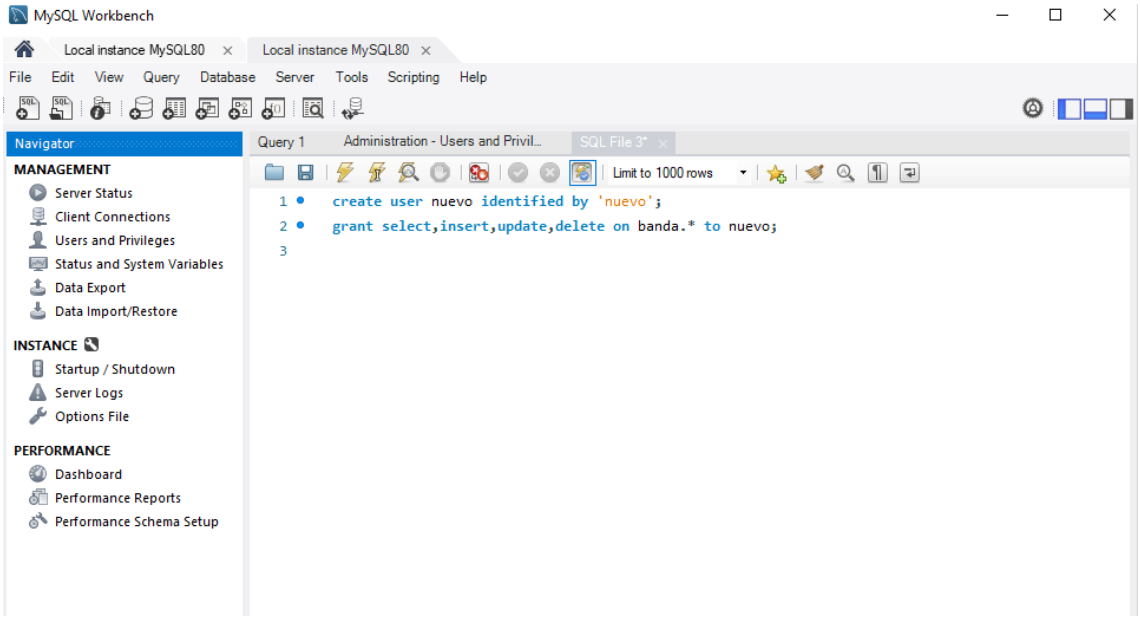
CONEXIÓN ENTRE JAVA Y MYSQL

Vamos a ver cómo utilizar la tecnología JDBC (Java Database Connectivity), un API de Java para poder acceder a nuestras BD, en concreto nosotros vamos a utilizarlo para acceder a mysql. En caso de querer acceder a SQL Server sería similar, solo cambiando la cadena de conexión y el driver correspondiente.

1. Pasos previos en mysql para trabajar con Java

- Creación de usuario en mysql para poder acceder a la BD de prueba (banda)

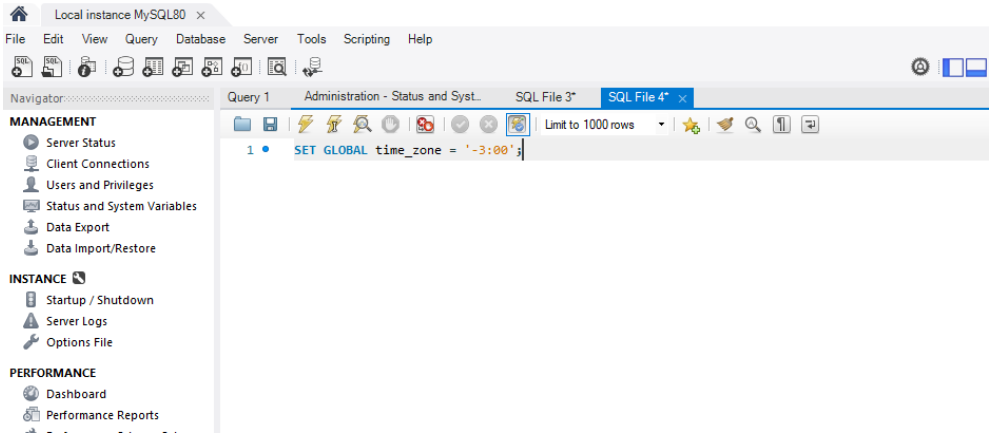


Creamos un usuario nuevo con permisos básicos de selección, inserción, borrado y modificación, los necesarios para realizar luego nuestro CRUD. El usuario creado de esta forma puede acceder desde cualquier equipo ya que se ha creado solo poniendo su nombre (así que por defecto utiliza nuevo@% que significa que puede abrir sesión desde cualquier máquina, si quisiésemos limitarlo pondríamos nuevo@ip o nombre de la máquina, p.e: nuevo@localhost solo podría iniciar sesión desde la misma máquina).

Siempre es interesante, por cuestiones de seguridad, utilizar un usuario con privilegios limitados y no el usuario root.

- Cambio de zona horaria

  Esto se realiza si al intentar conectar con la BD nos da un error de zona horaria, podríamos cambiar la URL pero para no hacerlo tenemos esta otra opción:



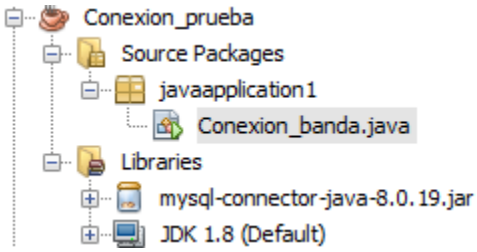
La otra opción es en Java utilizar este cambio en la URL de la conexión:

```
jdbc:mysql://localhost/db?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
```

2. Añadimos el driver en Java

Hemos de incluir el driver correspondiente en las librerías de nuestro proyecto, hemos de utilizar el .jar que está en C:\Program Files (x86)\MySQL\Connector J 8.0 y que es mysql-connector-java-8.0.19.jar (si lo habéis instalado cuando instalasteis mysql, sino id al centro de descargas de mysql y bajadlo).

Aquí vemos que en las Libraries de nuestro proyecto hemos de añadir dicho .jar (Botón derecho->Add JAR/Folder).



3. Establecemos la conexión

Ahora vamos a probar si conectamos adecuadamente, para ello vamos a necesitar el paquete **java.sql** que contiene todas las clases necesarias para el manejo de la BD (controladores, excepciones propias, interfaz con la aplicación).

Lo primero que se debe realizar para poder conectarse a una base de datos es cargar el driver encargado de esta función (por eso hemos registrado antes el controlador al indicar en las librerías el conector que íbamos a usar). Para ello se utiliza ***Class.forName*** pasándole el driver que queremos que se cargue.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Aquí va el correspondiente a mysql pero si quisiésemos utilizar una BD de por ejemplo SQL Server habría que indicar:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

Después hemos de crear la Conexión, para ello utilizamos el método getConnection de la clase DriverManager.

La conexión devuelta por el método ***DriverManager.getConnection*** es una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen nuestras sentencias SQL al controlador de la base de datos.

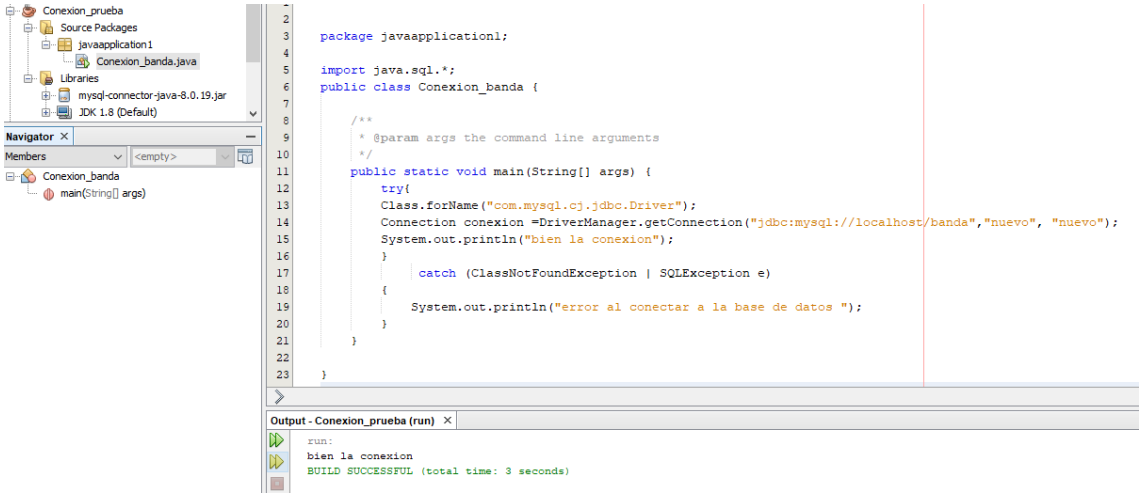
Fijáos en el formato: una URL, el usuario y su contraseña en la BD.

En la URL el formato es:

```
jdbc:subprotocolo:localizadorBD
```

- El subprotocolo indica el tipo de base de datos específico (en nuestro caso mysql)
- El localizador permite referenciar de forma única una BD :
    - Host y opcionalmente puerto (podríamos poner localhost:3306)
    - Nombre de la base de datos

  (en nuestro caso //localhost/banda porque está en la máquina local, si estuviese en otra máquina ahí pondríamos la ip o el nombre y después el nombre de la BD)



La conexión a la BD se hace con el método getConnection():

```
public static Connection getConnection(String url)
```

```
public static Connection getConnection(String url, String user, String password)
```

```
public static Connection getConnection(String url, Properties info)
```

Todos pueden lanzar la excepción SQLException por eso necesitamos utilizar un try-catch, de hecho si no lo hacemos nos salta un error. También hemos de tener en cuenta la ClassNotFoundException porque nos ocurre lo mismo con la carga del driver.

A partir de aquí el objeto conexión será el que utilicemos para referirnos a la BD y poder interactuar con ella.

En este ejemplo aparece en el propio main escrito el código para probar la conexión, pero es muy útil y debe hacerse así, crear una clase Conexión para ser usada desde nuestras aplicaciones e incluso puede contener distintos métodos para acceder a nuestra BD.

4. Realizar operaciones en la BD a través de JDBC

Echad un vistazo a
https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html#creating_statements

A la hora de realizar operaciones sobre la BD vía JDBC lo que intentamos es realizar consultas SQL embebidas en el código de Java, así que necesitamos un medio para poder escribir esas consultas. Para eso tenemos los Statements: son clases que nos van a permitir a través de sus métodos ejecutar las sentencias de sql en la bd. Tenemos 3 tipos: los que solo nos permiten consultas estáticas (Statement), los que nos permiten utilizar consultas con parámetros (PreparedStatement) y los que nos permiten utilizar llamadas a procedimientos almacenados (CallableStatement).

Todos los resultados que se obtengan de ejecutarlos se podrán recoger (cuando se devuelvan valores) en objetos de la clase ResultSet.

Así que los pasos serán, una vez creada la conexión:

- Crear el Statement asociado a la cadena de conexión
  ```
  Statement sentenza=conexion.createStatement();
  ```
- Crear la sentencia SQL e introducirla en el Statement , ejecutarla y recoger los resultados en un ResultSet.
  ```
  String cadena="SELECT * FROM musicos";
  ResultSet rs=sentenza.executeQuery(cadena);
  ```

Como las sentencias SQL pueden ser de consulta o de modificación en la BD, tendremos que tenerlo en cuenta también cuando ejecutemos dicha sentencia. Así, tendremos distintos métodos de ejecución:

- `executeQuery(String sql)`: Ejecución de consultas: SELECT y devuelve un objeto ResultSet
- `executeUpdate(String sql)`: Modificaciones en la BD: INSERT, UPDATE, DELETE y devuelve el número de columnas afectadas
- `execute(String sql)`: Ejecución de instrucciones que pueden devolver varios conjuntos de resultados. Requiere usar luego getResultSet() o getUpdateCount() para recuperar los resultados, y getMoreResults() para ver los siguientes resultados.

En cuanto al ResultSet generado hemos de tener acceso a los valores que se generan en la respuesta y tendremos que hacerlo con el método getX("campo"), donde X se refiere al tipo del dato en la tabla, por ejemplo si se trata de un INT en la BD será con getInt("edad"). Ver todas las posibilidades en el fichero *Material_AccesoBD_JDBC.pdf(pág. 12 y 13).*

4.1. Statement

En este ejemplo obtendríamos los músicos de la BD

```
package musica;

import java.sql.*;

public class Conexion {
      Connection conexion=null;
      public Conexion()
      {
        try{
        Class.forName("com.mysql.cj.jdbc.Driver");
        conexion =DriverManager.getConnection("jdbc:mysql://localhost/banda","nuevo", "nuevo");
         System.out.println("bien la conexion");
        }
            catch (ClassNotFoundException | SQLException e)
        {
            System.out.println("error al conectar a la base de datos ");
        }
      }
  public void seleccionar(Connection conexion,String mitabla)
    {
       try{

        String cadena="SELECT * FROM "+mitabla;
        Statement sentenza=conexion.createStatement();
        ResultSet rs=sentenza.executeQuery(cadena);

        while (rs.next())
        {
          String id = rs.getString("idmusico");
            String n = rs.getString("nombre");
            String a = rs.getString("apellidos");
            String d = rs.getString("direccion");
            System.out.println(id + "\t" + n +"\t" + a+ "\t" + d );
        }
       }
    catch (SQLException e)
        {
            System.out.println("error al conectar a la tabla");
        }
    }

}
```

Fijaos que ya lo he hecho en una clase aparte (ya no está en el main) y he aprovechado también para poner ahí uno de los métodos, aunque se podría poner en otra clase y así estaríamos evolucionando hacia el modelo-vista-controlador (MVC).

En el main asociado, por supuesto, la llamada sería sencilla:

```
package musica;

import java.sql.Connection;

public class MUSICA {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Conexion c=new Conexion();
        Connection miconexion=c.conexion;

        c.seleccionar(miconexion, "musicos");

    }

}
```

En caso de querer realizar una consulta en la que se realice una modificación en la BD, vamos a ver cómo utilizar el Statement, en este caso, con executeUpdate:

```
public static void engadir(Connection conexion)
{
    //lo ideal aquí es pasar como parámetros los valores a insertar, que ya hayan sido introducidos
    //en otra parte de la aplicación. Esto simplemente es para probar el funcionamiento del Statement
    //De esta forma se puede añadir, borrar y modificar simplemente cambiando la sentencia de la consulta
    try{
     Statement misentencia=conexion.createStatement();

     String id="19";
     String ape="rodriguez2";

         //aquí escribimos el insert con las variables que tienen los valores
       //  String consulta="insert into musicos values('19','rodriguez');
        String consulta="insert into musicos(idmusico,nombre) values('"+id+"','"+ape+"')";

        //Observad que hay que poner correctamente las comillas simples y dobles...

          //aquí ejecutamos el insert

          int filas=misentencia.executeUpdate(consulta);
          if (filas==0)
            System.out.println("Registro no insertado");
          else
            System.out.println("Músico añadido correctamente");

     }
    catch(SQLException e)
    {
       System.out.println("Error");
    }
}
```

En este ejemplo solo pasamos la conexión como parámetro, pero sería muy interesante pasar también los valores que van a ser introducidos en la BD.

4.2. PreparedStatement

Ahora vamos a ver cómo podríamos utilizar una sentencia con parámetros para utilizar los PreparedStatement.

En este caso es preparar el Statement llamando al método preparedStatement indicando con ? cada uno de los parámetros que luego vamos a darle su valor

```
PreparedStatement ps=conexion.prepareStatement("delete from musicos where
idmusico=?");
```

El siguiente paso es indicarle que variable tiene el valor que queremos que se asigne al parámetro y en este caso usamos el método setX con 2 parámetros el primero indica la posición del parámetro al que nos referimos(empieza a contar en 1) y el segundo la variable que tiene el valor que queremos que utilice.

```
ps.setString(1,clave);
```

Por último, tendremos que ejecutar la sentencia con el método executeUpdate y de nuevo deberemos controlar las posibles excepciones. Este método devuelve el número de elementos modificados, así que podemos comprobar si ha hecho alguna modificación o no

```
if (ps.executeUpdate()!=0) System.out.println("musico borrado");
```

Así nuestro nuevo método quedaría:

```
public static void borrarmusicos_con_parametros(Connection conexion,String clave)
{
     try{

    PreparedStatement ps=conexion.prepareStatement("delete from musicos where idmusico=?");
    ps.setString(1,clave);
    if (ps.executeUpdate()!=0)
        System.out.println("musico borrado");
     }
catch (SQLException e)
    {
        System.out.println("error al conectar a la tabla");
    }

}
```

Probad a realizar distintos métodos con cada uno de las posibles sentencias sql (Insert, delete, update).

4.3. Metadatos

Cuando ya hayáis avanzado suficiente, seguro que tendréis la necesidad de mostrar en una tabla la información, para ello utilizaréis un nuevo componente: JTable(en Swing Controls). Así ya podréis añadir una interfaz visual a los datos de vuestra BD y comenzar a construir vuestra aplicación. Cread en un nuevo Frame una tabla para poder unir GUI y persistencia de datos.

El siguiente código me permite rellenar un JTable utilizando los metadatos conseguidos a través de la ejecución de una consulta con un Statement. Tiene la ventaja de que me sirve para cualquier tabla.

```
public  static void seleccionar(Connection conexion,String mitabla,DefaultTableModel modelo)
{
   try{
       Statement sentenza=conexion.createStatement();
      try (ResultSet rs = sentenza.executeQuery("SELECT * FROM "+mitabla)) {
          ResultSetMetaData metaDatos = rs.getMetaData();

          // Se obtiene el número de columnas.
          int numeroColumnas = metaDatos.getColumnCount();

          // Se crea un array de etiquetas para rellenar
          Object[] etiquetas = new Object[numeroColumnas];

          // Se obtiene cada una de las etiquetas para cada columna
          for (int i = 0; i < numeroColumnas; i++)
          {
              // Nuevamente, para ResultSetMetaData la primera columna es la 1.
              etiquetas[i] = metaDatos.getColumnLabel(i + 1);
          }
          modelo.setColumnIdentifiers(etiquetas);

          while (rs.next()) {
              Object[] datosFila = new Object[modelo.getColumnCount()];
              for (int i = 0; i < modelo.getColumnCount(); i++)
                  datosFila[i] = rs.getObject(i + 1);
              modelo.addRow(datosFila);
          }
      }
       }
    catch (SQLException e)
       {
       System.out.println("error al crear la tabla ");
       }
}
```

Necesitamos poner en nuestra lista de imports:

```
import javax.swing.table.DefaultTableModel;
```

Y definir nuestro modelo de tabla(en este caso por defecto)con:

```
DefaultTableModel modelo = new DefaultTableModel();
```

antes de realizar la llamada (p.e.):

```
seleccionar(conexion,"musicos",modelo);
```

y para asignarlo a JTable:

```
tabla.setModel(modelo);
```

En caso de utilizar este código para recargar los datos de una JTable, sería interesante primero borrar las filas del modelo antiguas y luego añadir la nueva configuración. Para ello, necesitaremos borrar las filas del modelo antes de cargar las nuevas filas:

```
for (int i=modelo.getRowCount()-1;i>=0;i--)    modelo.removeRow(i);
```

Este código introducidlo al principio del método seleccionar (o por lo menos antes de añadir los datos de las filas).

Así quedaría:

```
public  static void seleccionar(Connection conexion,String mitabla,DefaultTableModel modelo)
{
   try{
        Statement sentenza=conexion.createStatement();
        for (int i=modelo.getRowCount()-1;i>=0;i--)  //para limpiar las filas de modelo
              modelo.removeRow(i);

       try (ResultSet rs = sentenza.executeQuery("SELECT * FROM "+mitabla)) {
           ResultSetMetaData metaDatos = rs.getMetaData();

           // Se obtiene el número de columnas.
           int numeroColumnas = metaDatos.getColumnCount();

           // Se crea un array de etiquetas para rellenar
           Object[] etiquetas = new Object[numeroColumnas];

           // Se obtiene cada una de las etiquetas para cada columna
           for (int i = 0; i < numeroColumnas; i++)
           {
               // Nuevamente, para ResultSetMetaData la primera columna es la 1.
               etiquetas[i] = metaDatos.getColumnLabel(i + 1);
           }
           modelo.setColumnIdentifiers(etiquetas);

           while (rs.next()) {
               Object[] datosFila = new Object[modelo.getColumnCount()];
               for (int i = 0; i < modelo.getColumnCount(); i++)
                   datosFila[i] = rs.getObject(i + 1);
               modelo.addRow(datosFila);
           }
       }
        }
    catch (SQLException e)
        {
        System.out.println("error al crear la tabla ");
        }
}
```

Además sería interesante si disponemos de JTable que al hacer click sobre ella, nos aparezcan los datos en JTexts, para ello podemos utilizar los métodos que nos proporciona la tabla(p.e):

```
private void tablaMouseClicked(java.awt.event.MouseEvent evt) {

    IdText.setText(tabla.getValueAt(tabla.getSelectedRow(), tabla.getSelectedColumn()).toString());
}
```

getSelectedRow() me devuelve la fila seleccionada y getSelectedColumn() la columna,

Con getValueAt podemos obtener el valor de una fila y una columna de la tabla en concreto.