UT2. Parte 3: Trabajando con JDBC y MySQL

En el apartado anterior trabajamos con una base de datos embebida, llamada SQLite. En este apartado vamos a hacerlo con un Sistema gestor de bases de datos relacionales, concretamente **MySQL**. Nos instalaremos un contenedor docker que contiene una imagen de MySQL

1. Demo 2.2. Instalación de MySQL mediante contenedores Docker en Linux

Docker es una herramienta que permite empaquetar una aplicación y sus dependencias en un contenedor virtual que se puede ejecutar en cualquier servidor Linux (también existen contenedores para otros sistemas operativos, pero Linux es el más habitual). Esto aporta flexibilidad y portabilidad a las aplicaciones, además de permitir aislar los servicios en el ordenador personal del desarrollador, sin tener que convivir ni molestarse entre ellos.

Antes de empezar, conviene aclarar 2 conceptos que vamos a utilizar, muy utilizados en Docker:

- Una ***imagen*** es un paquete ejecutable que incluye todo aquello necesario para ejecutar una aplicación: código, entorno de ejecución, librerías, variables de entorno y ficheros de configuración. Habitualmente se genera una imagen y esta no se modificará. Podemos entenderla como una imagen ISO: la cargamos, pero siempre es igual.
- Un ***contenedor***, por su parte, es una instancia de una imagen en ejecución: aquello que se crea cuando ponemos en marcha una imagen.

Paso 1. Instalación de Docker en Linux

1. Añadimos la clave oficial GPG (GNU Privacy Guard) (para garantizar la autenticidad e integridad del software de Docker):

   Nota: La **clave oficial GPG (GNU Privacy Guard)** es una clave criptográfica que Docker utiliza para firmar digitalmente su software (como paquetes, imágenes, o binarios). El propósito de esta clave es garantizar la **autenticidad** y **la integridad** del software, es decir, asegurar que:

   a. **El software proviene realmente de Docker, Inc.** y no ha sido alterado por un tercero.
   b. **El software no ha sido modificado o corrompido** durante la descarga.

```
$ sudo apt-get update

$ sudo apt-get install ca-certificates curl gnupg

$ sudo install -m 0755 -d /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /etc/apt/keyrings/docker.gpg

$ sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

2. Añadimos el repositorio para la descarga de Docker:

```
$ echo \

  "deb [arch="$(dpkg --print-architecture)"  signed-
     by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
     \

  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \

  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

3. Actualizamos los paquetes e instalamos:

```
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

4. Comprobamos que la instalación se ha realizado correctamente, ejecutando un contenedor denominado "Hola Mundo":

```
$ sudo docker run hello-world
```

```
alumno@ubuntu-alumno:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:88ec0acaa3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

alumno@ubuntu-alumno:~$ docker --version
Docker version 24.0.6, build ed223bc
```

5. Revisando la salida parece que haya habido un error, que no encuentra la imagen. Esto es debido a que la imagen no la tenemos en nuestro sistema. Se realiza un `pull` (descarga) de la imagen desde los repositorios de Docker (el nombre de la imagen es `hello-world:latest`). Las imágenes se identifican con su *nombre:versión*.

   La descarga de las imágenes se realiza la primera vez que se ejecutan. Docker mantiene en nuestro equipo aquellas imágenes que utilizamos. Una vez descargada, se ejecuta, por lo que podemos comprobar el mensaje ***Hello from Docker!***

   Podemos ver la versión de docker con el comando `docker –version`.

Nota. Verificación del estado de Docker:

- Para comprobar que el servicio de Docker está activo podemos ejecutar: `sudo systemctl status docker`
- Para parar el servicio de Docker: `sudo systemctl stop docker`
- Para iniciar el servicio de Docker: `sudo systemctl start docker`
- Para reiniciar el servicio de Docker: `sudo systemctl restart docker`

Paso 2: Docker y privilegios

Para ejecutar los comandos Docker en sistemas Linux, estamos obligados a realizarlo todo con privilegios de administrador, lo que implica el escribir «sudo» en cada comando. Para evitar esto, en la instalación de Docker se crea un grupo `docker` con permisos para realizar estas tareas. Añadiendo nuestro usuario personal a dicho grupo, evitamos tener que anteponer «sudo» en cada comando.

1. Agregamos el usuario al grupo Docker:

   ```
   sudo usermod -aG docker ${USER}
   ```

2. Al reiniciar la sesión del sistema, ya perteneceremos a dicho grupo y podremos ejecutar comandos Docker sin necesidad de escribir sudo.

Para saber más:

- Ver manual de instalación en la dirección https://docs.docker.com/engine/install/ubuntu/

Paso 3: Carga y ejecución de contenedor MySQL (versión 5.7)

Escribimos el siguiente comando para descargar una imagen de MySQL y ejecutarla como contenedor:

```
$ docker run --name mysql-5-container -e MYSQL_ROOT_PASSWORD=admin -p 3306:3306 -d mysql:5.7
```

```
alumno@ubuntu-alumno:~$ docker run --name mysql-5-container -e MYSQL_ROOT_PASSWORD=admin -p 3306:3306 -d mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
9ad776bc3934: Pull complete
9e4eda42c982: Pull complete
df6d882cf587: Pull complete
6c804e92b324: Pull complete
fd54ada0c48d: Pull complete
4ed8fb20ac8d: Pull complete
eec2b1bc5454: Pull complete
41c3423057b7: Pull complete
122b2c7b16c0: Pull complete
0d30e03d70e3: Pull complete
71c43898e898: Pull complete
Digest: sha256:4f9bfb0f7dd97739ceedb546b381534bb11e9b4abf013d6ad9ae6473fed66099
Status: Downloaded newer image for mysql:5.7
f7ae12ccc748269a44e7918c699ee02e0c20f60689837c239dedbacac3b96cfe
```

Opciones especificadas en el comando docker run ejecutado:

- **`--name`:** a la derecha colocaremos el nombre que queremos dar al contenedor: **mysql-5-container**
- **`-e`:** permite establecer variables de entorno, en este caso la password del usuario root (MYSQL_ROOT_PASSWORD) del contenedor MySQL Server a arrancar (asignamos la contraseña "**admin**").

- **`-p:`** equivalente a --publish, permite mapear puertos de la máquina (Ubuntu Linux) con puertos del contenedor (MySQL Server 5.7): mapeamos el puerto **3306**, que es el predeterminado en MySQL
- **`-d`**: equivalente a --detach. Ejecuta el contenedor *en segundo plano* y muestra en la terminal el ID de contenedor asignado.
- **`mysql:5.7`**: indicamos que queremos expresamente la versión 5.7 de MySQL. Para instalar la última versión sería `mysql:latest`.

Para comprobar que el contenedor está en funcionamiento, podemos ejecutar el comando **`docker ps`**, que muestra los contenedores en ejecución:

```
alumno@ubuntu-alumno:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS                                                  NAMES
f7ae12ccc748   mysql:5.7   "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql-5-container
alumno@ubuntu-alumno:~$
```

Nota: para arquitecturas ARM (por ejemplo Mac con chip M2) el comando a ejecutar sería el siguiente: `docker run --name mysql-5-container -e MYSQL_ROOT_PASSWORD=admin -p 3306:3306 -d arm64v8/mysql`

Comandos útiles Docker

Los contenedores son elementos efímeros, lo que quiere decir que los arrancamos, los utilizamos, los paramos y los podemos borrar también.

- **Parar un contenedor:** `docker stop mysql-5-container`
- **Arrancar un contenedor** que ya está en nuestro sistema: `docker start mysql-5-container`

```
alumno@pop-os:~$ docker start mysql-5-container
mysql-5-container
alumno@pop-os:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS         PORTS                                                  NAMES
d0b00897b050   mysql:5.7   "docker-entrypoint.s…"   2 hours ago   Up 3 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql-5-container
```

- **Borrar un contenedor:** `docker rm mysql-5-container`
- **Ver contenedores en ejecución:** `docker ps`

```
alumno@pop-os:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED      STATUS          PORTS                                                  NAMES
d0b00897b050   mysql:5.7   "docker-entrypoint.s…"   3 days ago   Up 16 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql-5-container
```

- **Ver todos los contenedores (en ejecución o parados):** `docker ps -a`

```
alumno@pop-os:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED              STATUS              PORTS                                                  NAMES
a7f9b4db1af9   postgres    "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp              srv-postgres
d0b00897b050   mysql:5.7   "docker-entrypoint.s…"   3 days ago           Up 23 minutes       0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql-5-container
alumno@pop-os:~$ docker stop srv-postgres
srv-postgres
alumno@pop-os:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS                     PORTS                                                  NAMES
a7f9b4db1af9   postgres    "docker-entrypoint.s…"   2 minutes ago   Exited (0) 4 seconds ago                                                  srv-postgres
d0b00897b050   mysql:5.7   "docker-entrypoint.s…"   3 days ago      Up 23 minutes              0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql-5-container
alumno@pop-os:~$
```

Nota:

- El comando `docker start` arranca un contenedor que ya tenemos creado.
- El comando `docker run` descarga y crea el contenedor en el caso de que no lo tengamos en nuestro sistema.

Conectar con el servidor MySQL a través de la terminal

1) Acceder al contenedor (servidor con MySQL 5.7 instalado): comando docker exec

`Ejecutamos el comando:`

```
$ docker exec -it mysql-5-container bash
```

Con este comando accedemos al sistema de archivos y al entorno interno del contenedor `mysql-5-container`:

- **`docker exec:`** ejecuta un comando dentro de un contenedor que ya está corriendo.
- **`-it:`**

  `-i`: activa el modo interactivo, permitiendo que se pueda escribir en el terminal.

  `-t`: asigna una terminal pseudo-TTY, para que el entorno sea más parecido al de una consola normal.
- **`mysql-5-container`:** es el nombre o ID del contenedor donde se quiere ejecutar el comando. En este caso, se llama `mysql-5-container`.
- **`bash:`** es el comando que se ejecuta dentro del contenedor. Aquí inicia una shell bash interactiva dentro del sistema de archivos del contenedor.

2) Acceder a la base de datos

Iniciamos sesión en el cliente de MySQL con el usuario root. La opción -p indica que hay que introducir una contraseña. Introduciremos la que pusimos con el comando "`docker run`" (admin)

```
bash-4.2# mysql -u root -p
```

```
alumno@ubuntu-alumno:~$ docker exec -it mysql-5-container bash
bash-4.2# mysql -u root -p
Enter password:  admin
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.43 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

3) Comandos en la consola MySQL

Ver bases de datos disponibles en el servidor:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

Salir de la consola de MySQL:

```
mysql> \q
Bye
bash-4.2#
```

Salir del contenedor y volver a nuestro sistema Ubuntu:

```
bash-4.2# exit
exit
alumno@pop-os:~$
```

Paso 4: Instalar MySQL Workbench

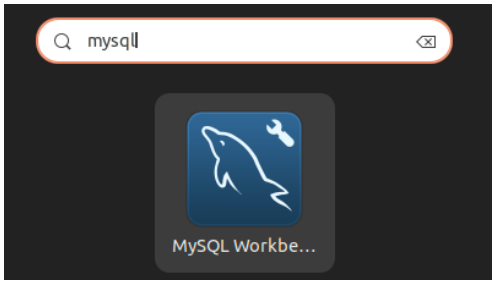
MySQL Workbench es una herramienta gráfica (GUI) desarrollada por Oracle que permite gestionar y administrar servidores de bases de datos MySQL de manera visual. La instalamos y ejecutamos después dos comandos para permitir el acceso de MySQL Workbench a las contraseñas almacenadas en el gestor de contraseñas del sistema y a las claves SSH para poder establecer conexiones seguras.

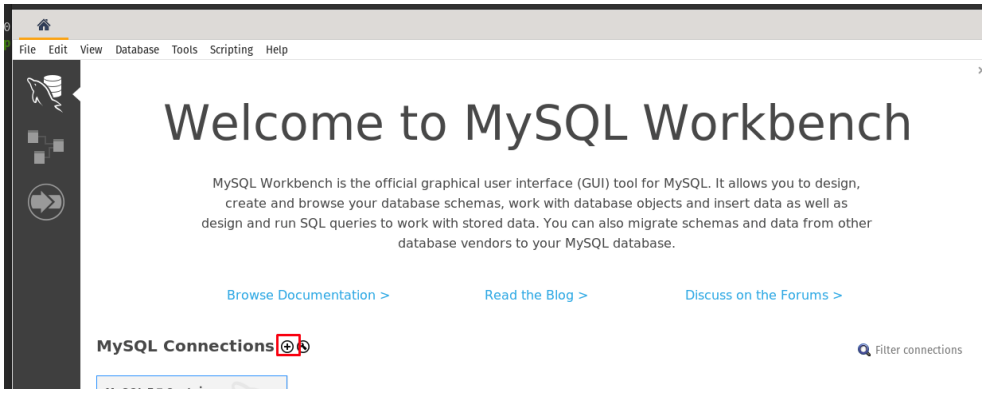
Ejecutamos:

```
$ snap install mysql-workbench-community

mysql-workbench-community 8.0.36 from Tonin Bolzan (tonybolzan) installed

$ snap connect mysql-workbench-community:password-manager-service

$ snap connect mysql-workbench-community:ssh-keys
```

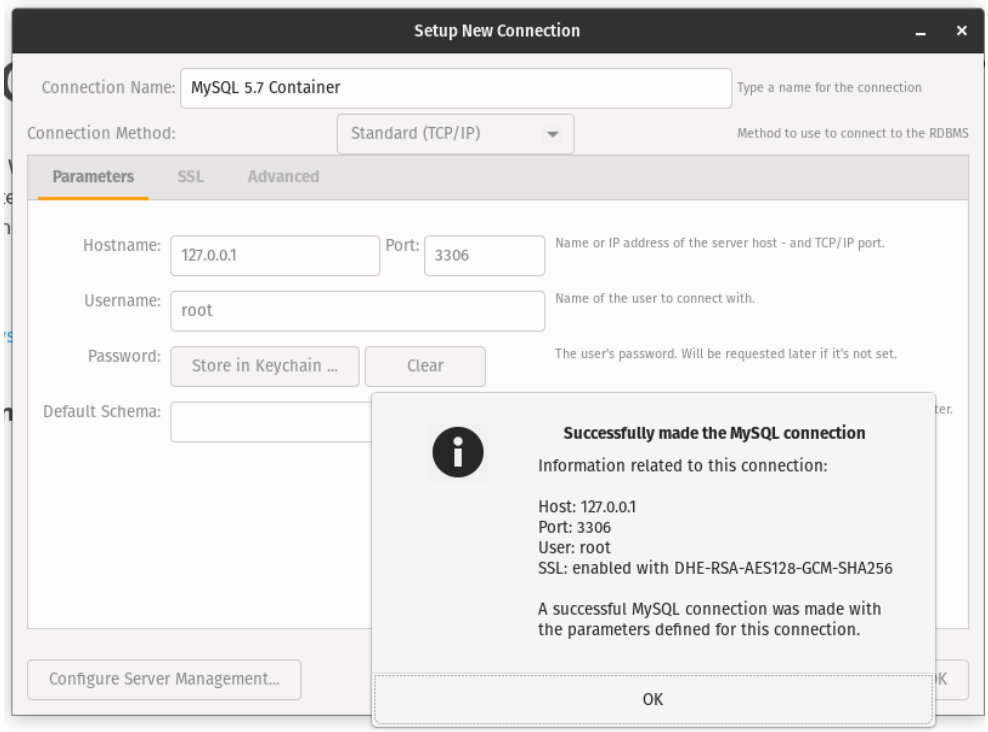
Configuramos una conexión desde MySQL Workbench

Vamos a configurar una conexión de nombre MySQL 5.7 Container, con usuario root y passwork admin.

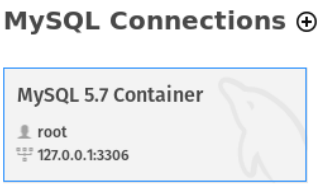


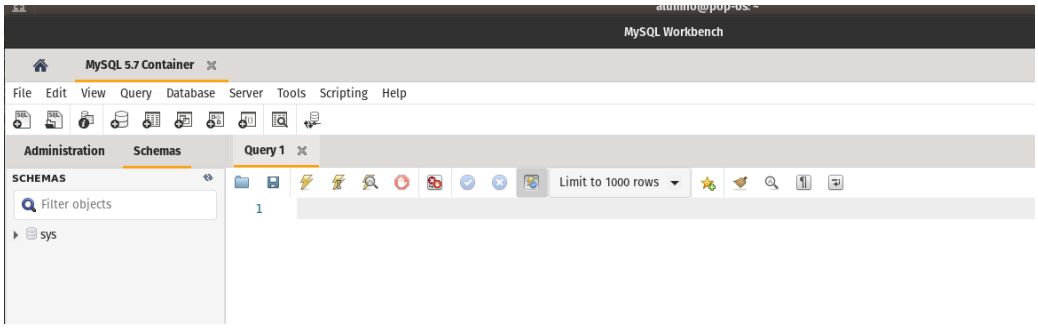


Hacemos click en el símbolo "+" al lado del texto "MySQL Connections", y en la ventana que se abre introducimos los datos de la conexión. Comprobamos la conexión con el botón "Test Connection".



Nos aparece un nuevo recuadro debajo de MySQL Connections para esta nueva conexión. Haciendo click en el recuadro accedemos a la misma.

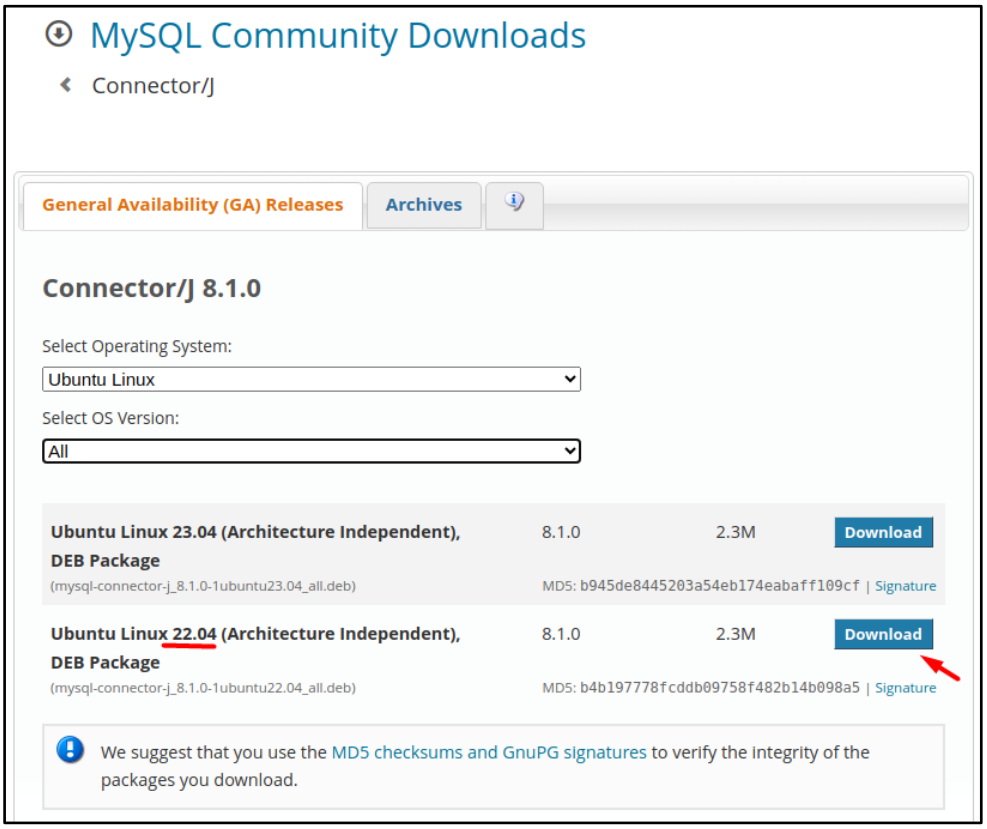




Nota: para sistemas con arquitectura ARM tendremos que acceder a la página de descarga de MySQL Workbench (https://dev.mysql.com/downloads/workbench/?os=33), descargar la aplicación e instalarla.

Paso 5: descarga del conector/driver JDBC para MySQL

Desde el navegador de la máquina virtual Linux, ir al enlace https://dev.mysql.com/downloads/connector/j/?os=26 y descargar el archivo:



Se almacenará un fichero con extensión `.deb` en Descargas.

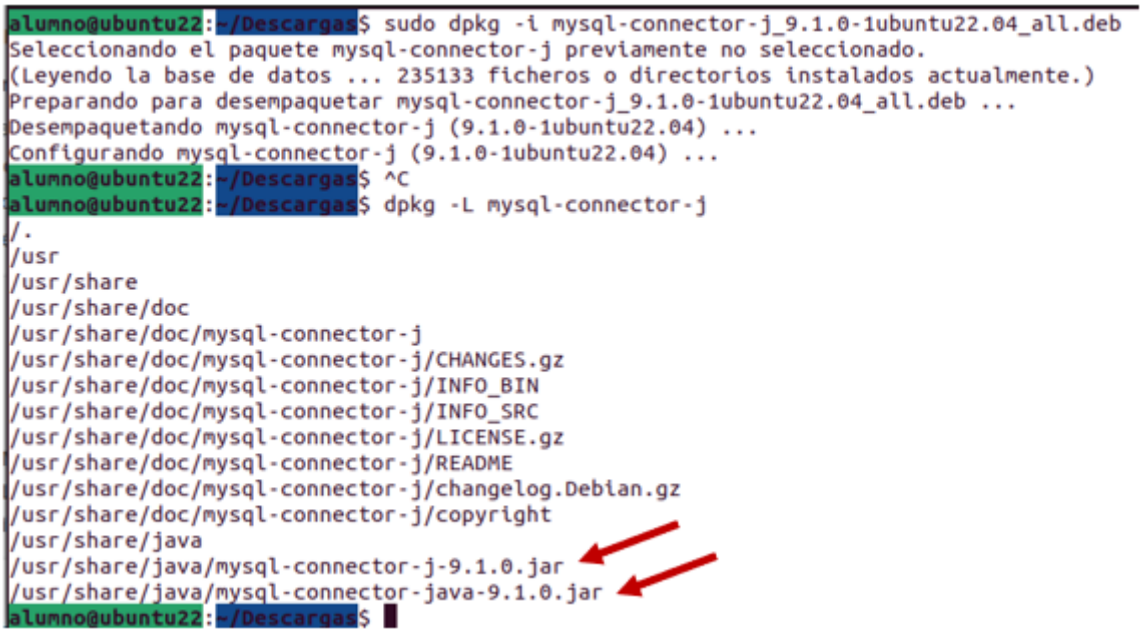
Abrimos una terminal y ejecutamos el paquete `.deb`:

```
$ sudo dpkg -i mysql-connector-j_9.1.0-1ubuntu22.04_all.deb
```

Para encontrar el archivo `.jar` tendremos que ejecutar el siguiente comando, que lista todos los archivos instalados por el paquete `mysql-connector-j` con sus rutas:

```
$ dpkg -L mysql-connector-j
```

Su ejecución mostrará lo siguiente:



Para tener el conector en un lugar más accesible, vamos a copiar cualquiera de los 2 marcados con una flecha roja en nuestra carpeta de descargas. Podemos hacerlo mediante el explorador de archivos o ejecutando el comando:

```
$ sudo cp /usr/share/java/mysql-connector-java-9.1.0.jar ~/Descargas/
```

2. Demo 2.3. Operaciones básicas con una base de Datos MySQL

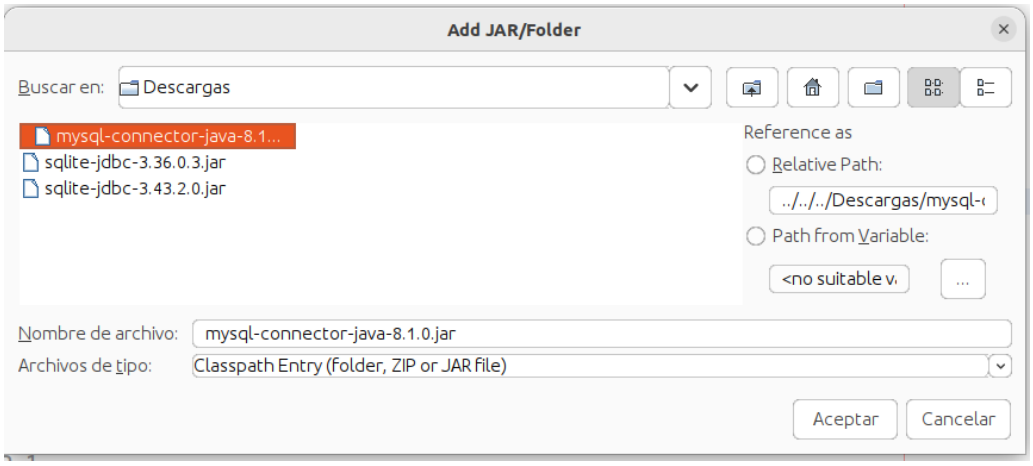
Vamos a realizar en MySQL lo mismo que hicimos en la Demo 2.1 con SQLite. Tendremos que realizar pequeñas modificaciones con respecto a la demo 2.1, ya que, a pesar de que el lenguaje SQL es un estándar, siempre hay pequeñas modificaciones entre diferentes sistemas de bases de datos. Algunos de estos cambios son debidos a los tipos de datos que manejan cada uno de estos sistemas.

A continuación se explican los pasos a seguir para crear el proyecto y prepararlo para la conexión a la base de datos, para después poder incluir el código Java de la Demo 2.1 adaptado a MySQL.

1. Crear un proyecto en NetBeans y preparar el entorno

I. **Creamos un proyecto** en NetBeans de nombre UT2_DEMO02_MySQL

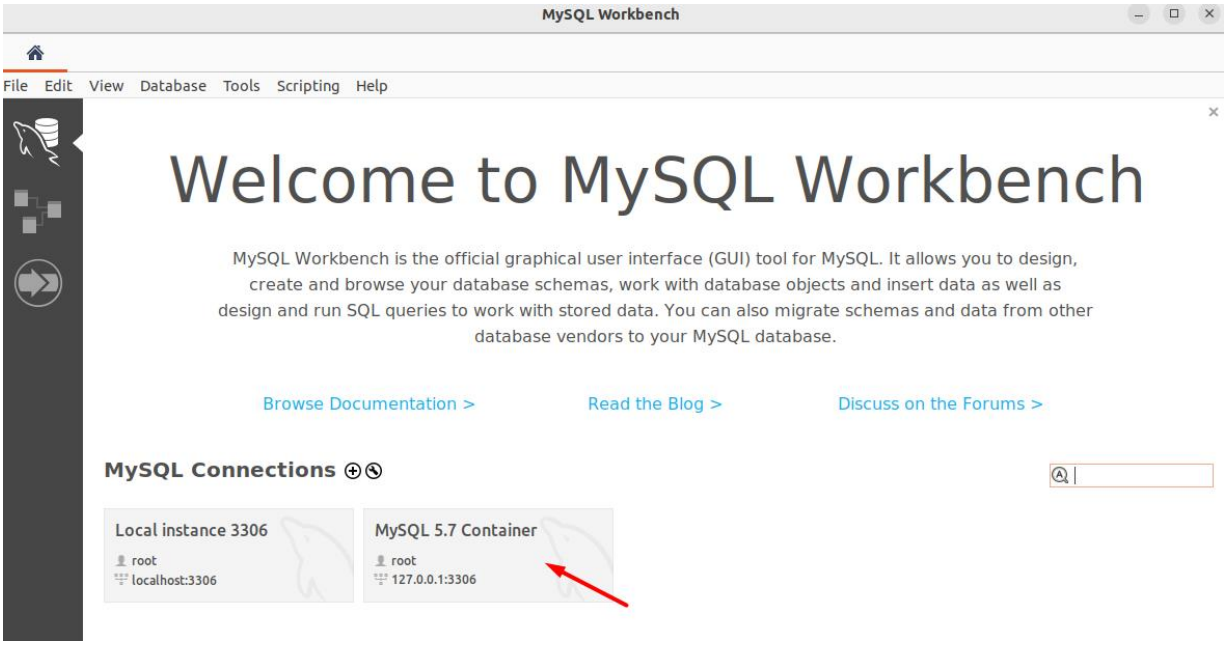
II. **Añadimos el driver al proyecto:** archivo .jar del conector JDBC para trabajar con MySQL, descargado anteriormente y guardado en Descargas:



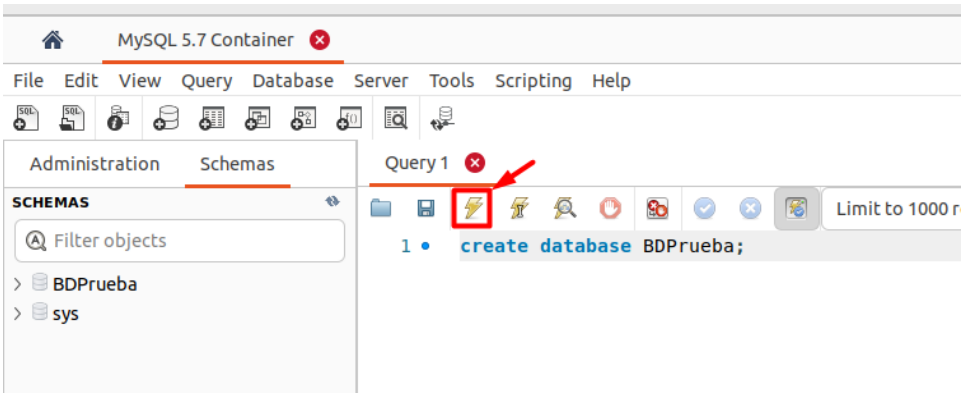
III. **Comprobamos que el contenedor Docker de MySQL está activo** (comando `docker ps`), y si no lo está tendremos que arrancarlo con el comando `docker start mysql-5-container`.

```
alumno@ubuntu-alumno:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
alumno@ubuntu-alumno:~$ docker start mysql-5-container
mysql-5-container
alumno@ubuntu-alumno:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS         PORTS                                                  N
AMES
f7ae12ccc748   mysql:5.7   "docker-entrypoint.s…"   2 hours ago   Up 2 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   m
ysql-5-container
```

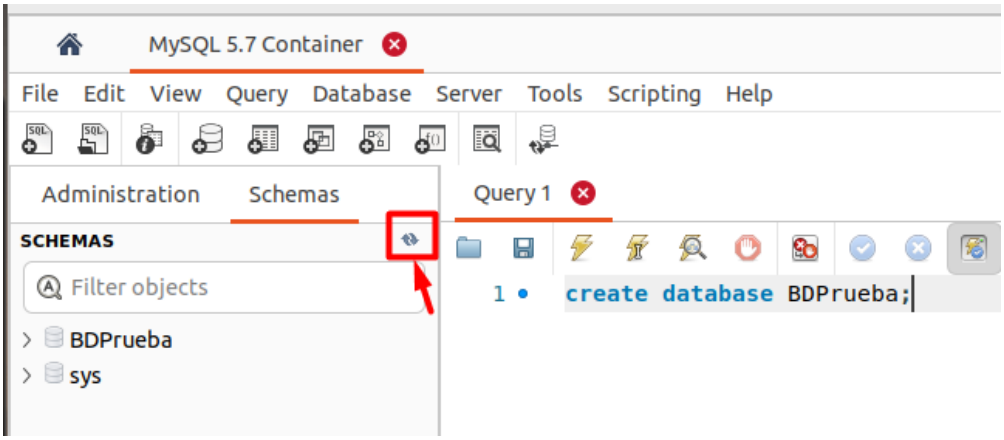
IV. **Creamos la base de datos en el servidor (contenedor Docker) MySQL**. Para ello, abrimos MySQL Workbench, nos conectamos utilizando la conexión creada anteriormente (MySQL 5.7 Container)



Seleccionamos la pestaña Schemas del panel de la izquierda y ejecutamos lo siguiente:



Para que se vea nuestra nueva base de datos creada tenemos que pinchar en el botón Refresh:



---

2. Conexión con la base de datos MySQL

Cadena de conexión

Recordemos que para establecer una conexión a una base de datos con JDBC necesitamos una «***cadena de conexión***», que es un *String* cuyo formato genérico es:

```
jdbc:driver://servidor:puerto/nombre_base_datos
```

Habitualmente MySQL escucha por el puerto 3306, aunque dichos puertos pueden modificarse por decisiones de seguridad o por tener varios servidores ejecutándose a la vez (no pueden usar el mismo puerto).

En MySQL, un ejemplo básico de cadena de conexión a una base de datos MySQL en la máquina local sería:

```
String connectionUrl = "jdbc:mysql://localhost:3306";
```

- Si tuviéramos un servidor MySQL en otra máquina con otra dirección IP, bastaría con sustituir `localhost` por la dirección (o nombre dns) del servidor, por ejemplo:

  ```
  jdbc:mysql://192.168.1.254:3306
  ```

- En la cadena de conexión, de manera obligatoria tenemos que indicar el *host (servidor)* y el *puerto*. Podemos además especificar una *base de datos* concreta que, además, quedaría marcada como activa.:

  ```
  String connectionUrl = "jdbc:mysql://localhost:3306/BDPrueba";
  ```

La conexión a la base de datos se establece con el método getConnection de la clase DriverManager. Recordemos que este método tiene varias versiones (url representa la cadena de conexión a la base de datos):

- **static Connection getConnection(String url)**
- **static Connection getConnection(String url, Properties info):** algunos parámetros están especificados en la URL y otros en un objeto de propiedades (Properties).
- **static Connection getConnection(String url, String user, String password):** los datos de usuario y contraseña se suministran en dos parámetros adicionales.

Usuario y contraseña

En MySQL, a diferencia de SQLite, tenemos que autenticarnos para acceder a la base de datos. Una manera de hacerlo sería utilizar la versión del método getConnection que recibe como parámetros el usuario y la password de conexión:

```
String connectionUrl = "jdbc:mysql://localhost:3306";

Connection conn = DriverManager.getConnection(connectionUrl, "root", "admin");
```

Nota:

Como hemos indicado en la creación de nuestro contenedor Docker de MySQL que "mapee" lo que le llegue por el puerto 3306 del localhost (nuestra máquina Ubuntu) con el puerto 3306 del contenedor, la conexión con nuestro servidor debería realizarse de forma correcta.

Añadir más parámetros en la cadena de conexión

Se pueden añadir propiedades o parámetros adicionales a la conexión dentro de la cadena de conexión. Deben indicarse siguiendo el protocolo HTTP, incluyendo un símbolo de cierre de interrogación (`?`), seguido de una lista de pares `atributo=valor` separados por el símbolo `&`.

Ejemplo 1: la cadena de conexión incluye los valores de user y password

```
String connectionUrl =
 "jdbc:mysql://localhost:3306/BDPrueba?user=root&password=admin";

Connection conn = DriverManager.getConnection(connectionUrl);
```

Ejemplo 2: Se añaden dos parámetros más a la cadena de conexión para habilitar Unicode y especificar codificación UTF-8.

```
String connectionUrl =
  "jdbc:mysql://localhost:3306/BDPrueba?user=root&password=admin&useUnicode=true&c
  haracterEncoding=UTF-8";

Connection conn = DriverManager.getConnection(connectionUrl);
```

Formas más seguras de indicar al programa los datos de acceso a la base de datos (user, password, nombre de la base de datos)

Para evitar que viajen en la cadena de conexión datos sensibles "*hard-codeados*", podemos hacer varias cosas:

a. Introducir *user*, *password* y *nombre de la base de datos* como argumentos de entrada a nuestro programa.
b. Utilizar un objeto de la clase `java.util.Properties`. (ver https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html)
c. Leer dichos datos de un fichero.

---

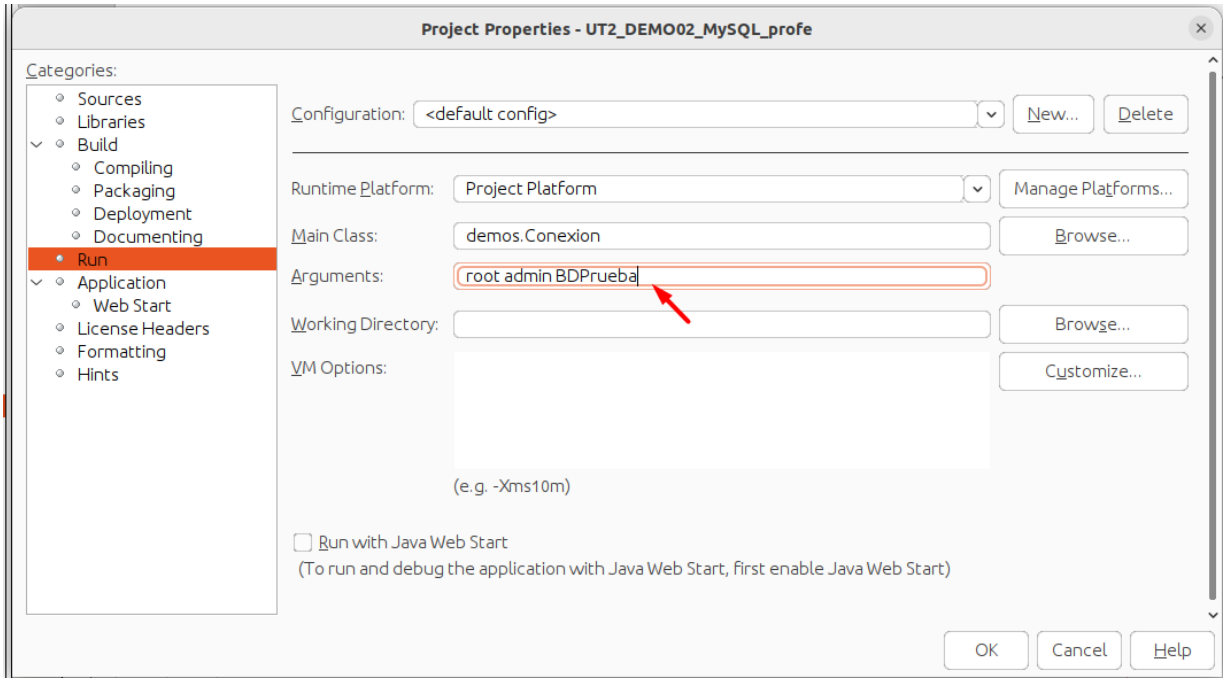
a) Pasando los datos como argumentos al main

El método main recibe 3 argumentos: el nombre de usuario, la contraseña y la base de datos. La cadena de conexión se construye con los valores de estos argumentos.

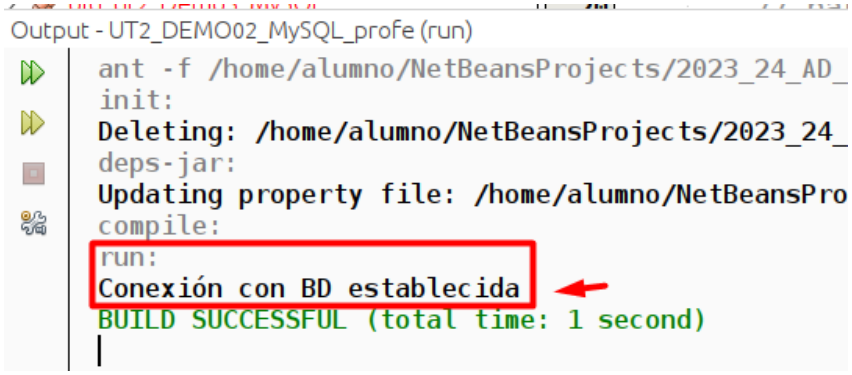
Ejemplo:

```
public class Demo2_3_MySQL_Opcion_A {

    public static void main(String[] args) {

        // Comprobamos si el usuario ha introducido 3 parámetros.
        // No vamos a comprobar si los 3 parámetros se corresponden
        // con lo que tienen que representar (user, pwd, nombre bdd).
        if (args.length != 3) {
            System.out.println("Número incorrecto de parámetros");
        } else {
            String usuario = args[0];
            String pwd = args[1];
            String dbName = args[2];

            //Cadena de conexión. Para ususrio=root, password=admin y dbName=BDPrueba quedaría:
            //"jdbc:mysql://localhost:3306/BDPrueba?user=root&password=admin&useUnicode=true&characterEncoding=UTF-8"
            String connectionUrl = "jdbc:mysql://localhost:3306/"
                    + dbName
                    + "?user=" + usuario
                    + "&password=" + pwd
                    + "&useUnicode=true&characterEncoding=UTF-8";

            try (Connection conexion = DriverManager.getConnection(connectionUrl)) {

                Class.forName("com.mysql.cj.jdbc.Driver");  // Desde Java 6 no hace falta
                System.out.println("Conexión con BD establecida");

            } catch (ClassNotFoundException ex) {
                System.out.println("ClassNotFoundException generada: " + ex.getMessage());
            } catch (SQLException ex) {
                System.out.println("SQLException generada: " + ex.getMessage());
            }//fin try-catch
        }// fin if-else
    } // Fin método main
}//Fin clase Demo2_3_MySQL_Opcion_A
```

En NetBeans, para especificar argumentos en la ejecución de un programa podemos hacer lo siguiente. Pinchamos con el botón derecho en el proyecto y seleccionamos 'Properties'. En la ventana que aparece, vamos al menú Run y rellenamos el campo Arguments. También tendremos que asegurarnos de que la "Main Class" es la que queremos ejecutar.



Pinchamos OK y, al ejecutar, debería aparecer el mensaje de "Conexión con BD establecida":



b) Con java.util.Properties y setProperty()

- La clase **`java.util.Properties`** permite el almacenamiento de pares clave-valor como cadenas. Todas las claves (keys) y valores son de tipo String.
- El método **`setProperty(String key, String value)`** permite definir una clave y su valor.
- En nuestro programa creamos en un objeto de la clase Properties y usamos setProperty para asignar valores las propiedades "user", "password", "useUnicode" y "characterEncoding".
- Para establecer la conexión utilizamos la variante `static Connection getConnection(String url, Properties info)` de método getConnection, al que pasamos como segundo argumento el objeto properties creado.

Ejemplo:

```
public class Demo2_3_MySQL_Opcion_B {
    public static void main(String[] args) {

    String connectionUrl = "jdbc:mysql://localhost:3306/BDPrueba";


        // Creamos objeto Properties y le añadimos valores
        Properties info = new Properties();

        // Cabecera del método setProperty:
        // public synchronized Object setProperty(String key, String value)
        //La sincronización garantiza que las propiedades se manejen de manera segura en entornos multihilo
        info.setProperty("user", "root");
        info.setProperty("password", "admin");
        info.setProperty("useUnicode", "true");
        info.setProperty("characterEncoding", "UTF-8");

        // Utilizamos el método getConnection añadiendo el objeto info creado
        try (Connection conexion = DriverManager.getConnection(connectionUrl, info)) {

            System.out.println("Conexión con BD establecida");

        //} catch (ClassNotFoundException ex) {  //No se genera si no usamos Class.forName
        //    System.out.println("ClassNotFoundException generada: " + ex.getMessage());
        } catch (SQLException ex) {
            System.out.println("SQLException generada: " + ex.getMessage());
        }

    } // Fin método main
}
```

Algunas de las propiedades comunes que se suelen configurar, aparte de las utilizadas en el ejemplo, incluyen:

- `useSSL`: Un valor booleano que indica si la conexión debe usar SSL. Por ejemplo, `props.setProperty("useSSL", "false")`.
- `autoReconnect`: Un valor booleano que indica si se debe intentar reconectar automáticamente en caso de que la conexión se pierda. Por ejemplo, `props.setProperty("autoReconnect", "true")`.
- `serverTimezone`: El timezone del servidor MySQL. Por ejemplo, `props.setProperty("serverTimezone", "UTC")`.
- `allowPublicKeyRetrieval`: Un valor booleano que indica si se debe permitir la recuperación de claves públicas. Por ejemplo, `props.setProperty("allowPublicKeyRetrieval", "true")`.
- `verifyServerCertificate`: Un valor booleano que indica si se debe verificar el certificado del servidor cuando se utiliza SSL. Por ejemplo, `props.setProperty("verifyServerCertificate", "false")`.
- `requireSSL`: Un valor booleano que indica si se debe requerir una conexión SSL. Por ejemplo, `props.setProperty("requireSSL", "true")`.

---

c) *Con java.util.Properties y leyendo los datos de un fichero ".properties"*

- En este apartado, vamos a utilizar la clase `Properties` y un fichero con extensión `.properties`.
- Nos creamos un objeto de la clase Properties y utilizamos el siguiente método de esta clase, que lee una lista de propiedades (pares clave-valor) de un flujo de caracteres de entrada en un formato sencillo orientado a líneas.:

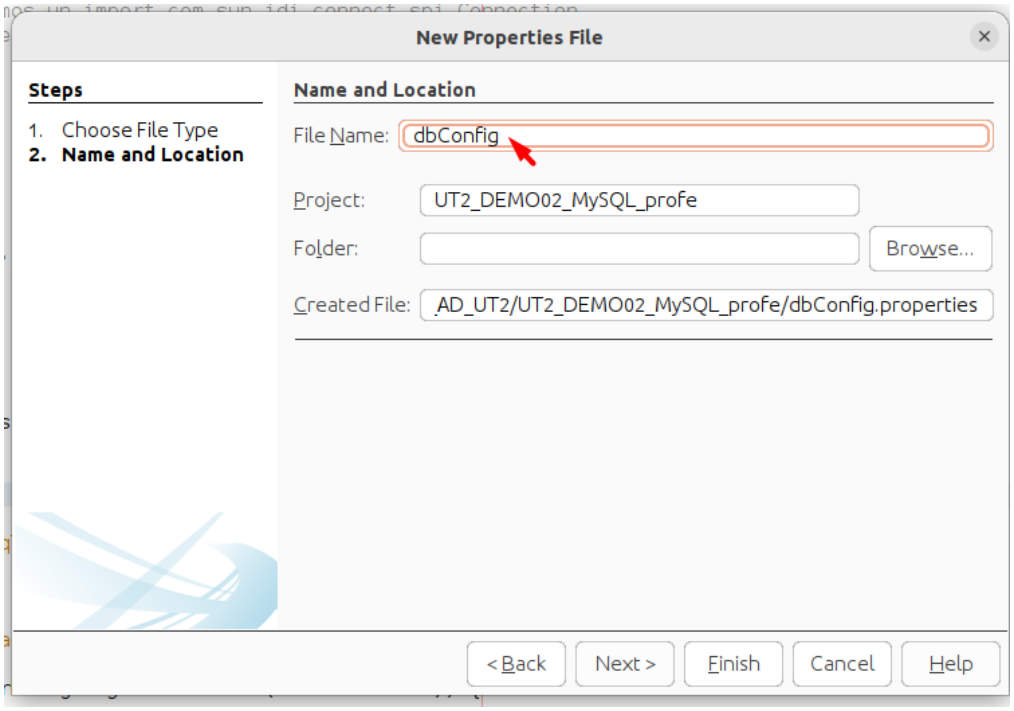
```
public void load(Reader reader)throws IOException
```

- El archivo al que se asocia el Reader contiene en cada línea un par clave-valor, con el formato `clave=valor`.Definiremos en este archivo todos los parámetros que queremos incluir en la cadena de conexión a la base de datos.

Ejemplo:

```java
public class Demo2_3_MySQL_Opcion_C {
    public static void main(String[] args) {

        Properties config = new Properties();

        // 1. Leemos el fichero de configuración
        try {
            //cargamos el archivo Properties. En este ejemplo está en src/Demo02/dbConfig.properties
            config.load(new BufferedReader(new FileReader("src/Demo02/dbConfig.properties")));
        } catch (IOException ex) {
            System.err.println("IOException generada. " + ex.getMessage());
        }

        String usuario = config.getProperty("user");
        String pwd = config.getProperty("password");
        String dbName = config.getProperty("dbName");

        // 2. Creamos la cadena de conexión.Podríamos leer también del fichero el nombre/ip del
        // servidor MySQL, el puerto, etc (toda la info que tengamos en el fichero
        // dbConfig.properties)y después construir la cadena de conexión
        String connectionUrl = "jdbc:mysql://localhost:3306/"
                + dbName
                + "?user=" + usuario
                + "&password=" + pwd
                + "&useUnicode=true&characterEncoding=UTF-8";

        //3. Establecemos la conexión a la base de datos
        try (Connection conexion = DriverManager.getConnection(connectionUrl)) {

            System.out.println("Conexión con BD establecida");

        } catch (SQLException ex) {
            System.out.println("SQLException generada: " + ex.getMessage());
        }
    } // Fin método main
}
```

Para que funcione, tenemos que crear un fichero `.properties` para el paquete (pinchar botón derecho en el nombre del paquete, New > Properties file). Lo llamamos `dbConfig` (se añadirá por defecto la extensión `.properties`. Si no especificamos Folder, se crea en la carpeta del proyecto.



Incluimos los siguientes datos en dbConfig.properties:

```
# Configuración de la conexión al servidor
user = root
password = admin
dbName = BDPrueba
useUnicode = true
characterEncoding = UTF-8
```

Ahora ya podríamos probar y ver que, efectivamente, la conexión se establece correctamente:

```
compile-single:
run-single:
Conexión con BD establecida
BUILD SUCCESSFUL (total time: 2 seconds)
```

- **Ejercicio:** hacer los cambios necesarios para que los valores de servidor y puerto de la cadena de conexión también se lean del archivo dbConfig.properties.

3. Tipos de datos en MySQL

Numéricos:

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)

Data type	Description
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Caracteres:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1

Data type	Description
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Fechas y tiempo:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time

TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Ejercicio: Actividad UT2_02, apartado 1

Crear un proyecto en NetBeans, de nombre **UT2_ACT02_MYSQL_ALUMNOS**. Modificar la aplicación de la actividad UT2_01 para que funcione con MySQL. Los parámetros de la conexión a la base de datos (servidor, puerto, usuario, contraseña, nombre de la base de datos, el uso de Unicode y codificación UTF-8) se leen de un fichero. Hacer dos versiones:

a) los datos de la conexión están en un archivo de texto con extensión ".properties", del que se leen y se recuperan mediante un objeto Properties
b) los datos se leen de un fichero binario (previamente debes crearte el fichero con los datos de la conexión).