DESARROLLO DE INTERFACES
2º DAM

I.E.S. POLITÉCNICO H.LANZ
JOSÉ MARÍA MOLINA

TEMA 2-3 – CRUD BBDD

2-3 CRUD BBDD

Vamos a crear una APP de gestión de datos básica donde se puedan realizar todas las operaciones básicas (CRUD). Esa APP tendrá que comunicar variables (o no) según su diseño (según si tenemos 1 o más ventanas).

- ✓ 1 ALTERNATIVAS DE DISEÑO
- ✓ 2 ALTERNATIVAS GESTIÓN DE DATOS
- ✓ 3 MARIADB VS MYSQL
- ✓ 4 SERVICIO EN DOCKER
- ✓ 5 CRUD

1 – ALTERNATIVAS DE DISEÑO

- ✓ Conforme una interfaz va creciendo va necesitando más ventanas y por tanto vamos a necesitar COMUNICARLAS. Una APP de BBDD podría necesitar muchas ventanas. En este punto tenemos distintas alternativas:

- **A) USANDO UNA ÚNICA VENTANA(SCENE) EN 1 FXML :** En este caso tenemos aquellas APP que se organizan SIN VENTANAS EMERGENTES, utilizando **STACKPANES** y/o **TABPANES** para toda la gestión.

- Ventajas: no necesita paso de variables.

- Inconvenientes: una única clase con mucho código, emplear bien el espacio sobrante, etc…

1 – ALTERNATIVAS DE DISEÑO

- **B) USANDO DISTINTAS VENTANAS SCENES (VARIOS FXML):** Se pueden tener mezcla de **STACKPANES, TABPANES** pero alguna o varias ventanas se abre de forma emergente.

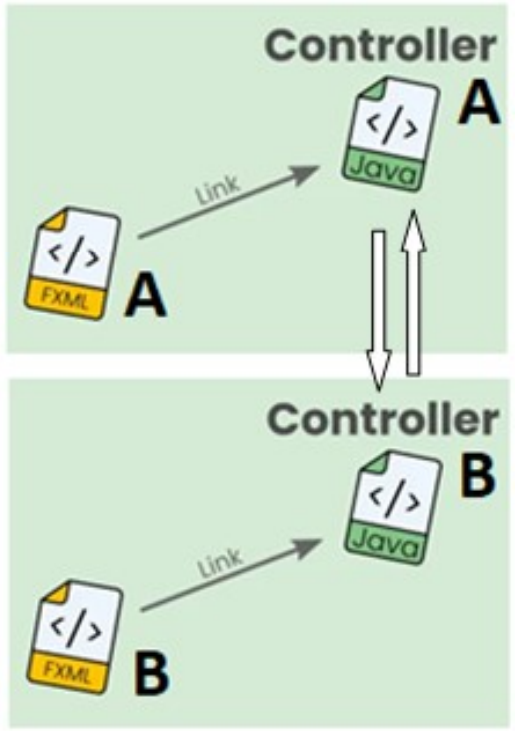
- Ventajas: mejor organizado tanto por el código como por el espacio.

- Inconvenientes: necesita **paso de variables** y tener varios FXML (uno por ventana) y además, una de las variables a pasar será el **conector** de la BBDD.

- Por ejemplo: Tengo un listado de **Productos** y quiero **EDITAR** uno en concreto en una ventana emergente. **¿¿Qué tenemos que hacer???**

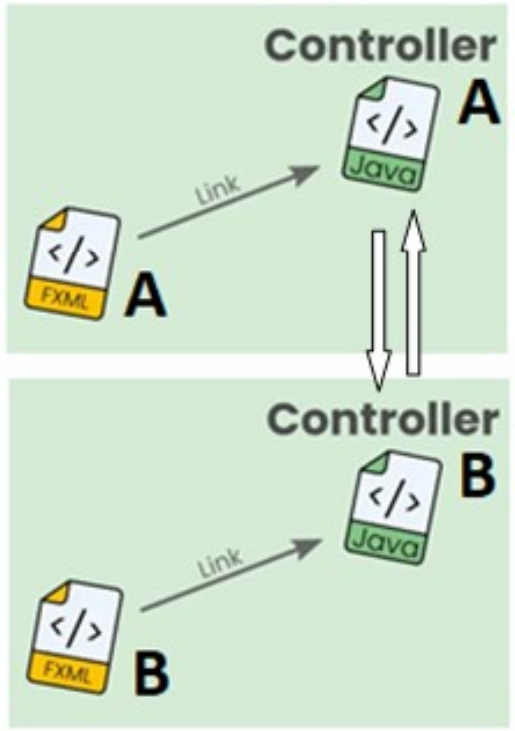
1 – ALTERNATIVAS DE DISEÑO

- **PASO DE VARIABLES:** Hay varias formas de hacer esto, cada una con sus ventajas e inconvenientes. Lo vemos en: **[javafx-ejemplos-pasovariables]**
- **Comunicar controladores [v1 y v2]**
  - Ventaja: es lo más sencillo puesto que es una forma "natural" de comunicar clases en java.
  - Inconveniente: hay que hacer métodos públicos (va en contra del encapsulado)



1 – ALTERNATIVAS DE DISEÑO

- ✓ **Comunicar controladores [v1]**
  - Problema: Crea tantas ventanas como veces le demos al botón (en el caso de no hacerlo modal)
- **Comunicar controladores [v2]**
  - Se reorganiza el código para evitar el problema anterior.



1 – ALTERNATIVAS DE DISEÑO

- ✓ **PASO DE VARIABLES:** Hay varias formas de hacer esto, cada una con sus ventajas e inconvenientes. Lo vemos en **[javafx-ejemplos-pasovariables]**
- ✓ **Mediante getUserData y setUserData [v3]**. Son dos métodos que "anclan" datos de cualquier tipo (objeto) a cualquier tipo de NODO o a incluso al propio STAGE.

– Inconveniente: hace el código un poco más enrevesado.

– Ventajas:

- No hay que hacer métodos públicos
- Muy versátil

1 – ALTERNATIVAS DE DISEÑO

- ✓ **ACCESO A VARIABLES:** Hay una serie de funciones muy útiles para acceder a variables en la jerarquía Stage->Scene->Nodo y así poder ver sus propiedades, acceder a la escena, a los nodos..
- ✓ **Acceso al Scene o al Stage**: se hace a partir de **cualquier componente** de la escena, siempre y cuando NO estemos en la función initialize (porque aún no se ha mostrado nada y por tanto→nada es accesible). Ejemplos de uso: acceso para añadir un nodo por código al scene o para ver atributos de ventana (stage), para acceder a etiquetas FXML, etc etc..
  - Scene miScene = (Scene) this.cajatexto.getScene();
  - Stage miStage = (Stage) this.cajatexto.getScene().getWindow();
- ✓ **Acceso al componente por etiqueta FXML**: se utiliza cuando quiero acceder a un elemento de un FXML que no es el que tengo cargado en el controlador actual
  - TextField t2 = (TextField) scene.lookup("**#cajaTexto2**");//Acceso por ID

2 – ALTERNATIVAS GESTIÓN DE DATOS

ALTERNATIVAS GESTIÓN DATOS

- ✓ **Y ahora, dónde guardo los datos de nuestra APP??** (ACCESO A DATOS)
- – **Ficheros locales**: XML, JSON, BINARIOS,TXT etc…
- – **BBDD Local**
  - Sin SGBDR: SQLITE
  - Tecnologías SQL con SGBDR: MaríaDB, MySQL, SQLServer, etc..
  - Tecnologías NOSQL: Firebase, MongoDB, etc..
- – **BBDD Local pero virtualizada**: VMWARE/VirtualBOX o DOCKER/DOCKER WSL
- – **BBDD Remota**
- – **BBDD vía ApiREST**

3 – MARIADB VS MYSQL

MARIADB VS MYSQL

✓ Aunque MariaDB es una bifurcación de MySQL, estos dos sistemas de gestión de bases de datos siguen siendo bastante diferentes:

– MariaDB tiene licencia GPL mientras que MySQL tiene un enfoque de doble licencia (desde que lo compró ORACLE). No es libre (no accedes a todas las características) pero puedes comprar todo, pero en ningún caso es de código abierto.

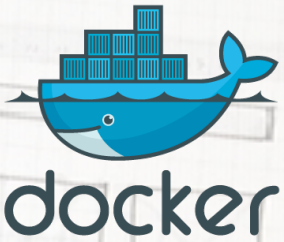
– MariaDB soporta muchos motores de almacenamiento diferentes.

– En muchos escenarios, MariaDB ofrece un mejor rendimiento.

– Usan MySQL: YouTube, GitHub, Spotify, Netflix y la NASA, HP, Amazon Web Services, IBM, Microsoft y Zapier.

– Usan MariaDB: Wikipedia, Google, CentOS y Verizon.

4 - SERVICIO EN DOCKER



✓ MONTANDO SERVICIOS EN DOCKER

– Ya sabemos trabajar con Docker, ahora solo falta instalar los contenedores que nos hagan falta. Alternativas??:

- **A) Contenedor Mariadb** + enlace a **Phpmyadmin.**

– Build.gradle: implementation 'org.mariadb.jdbc:mariadb-java-client:3.1.2‘

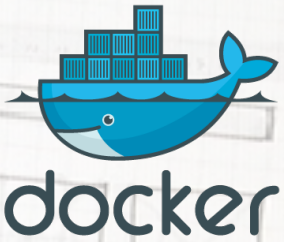
– Driver de conexión conn = DriverManager.getConnection("jdbc:**mariadb**://localhost:3306/libreria", "root", "rootpwd");

- **B) Contenedor MySQL**

– Build.gradle: implementation 'mysql:mysql-connector-java:8.0.32‘

– Driver de conexión: conn = DriverManager.getConnection("jdbc:**mysq**l://localhost:3306/libreria", "admin", "jmb6M9xC4OPR")

4 - SERVICIO EN DOCKER



✓ MONTANDO SERVICIOS EN DOCKER

– Ya sabemos trabajar con Docker, ahora solo falta instalar los contenedores que nos hagan falta. Vemos 3 alternativas:

- **C) Contenedor LAMP** (exponiendo además el puerto 3306)

sudo docker run --name LAMP --restart=always -p "80:80" -p "3306:3306" -v /home/usuario/app:/app mattrayner/lamp:latest-1804

– Build.gradle: podemos usar cualquiera de los dos implements (cualquier conector)

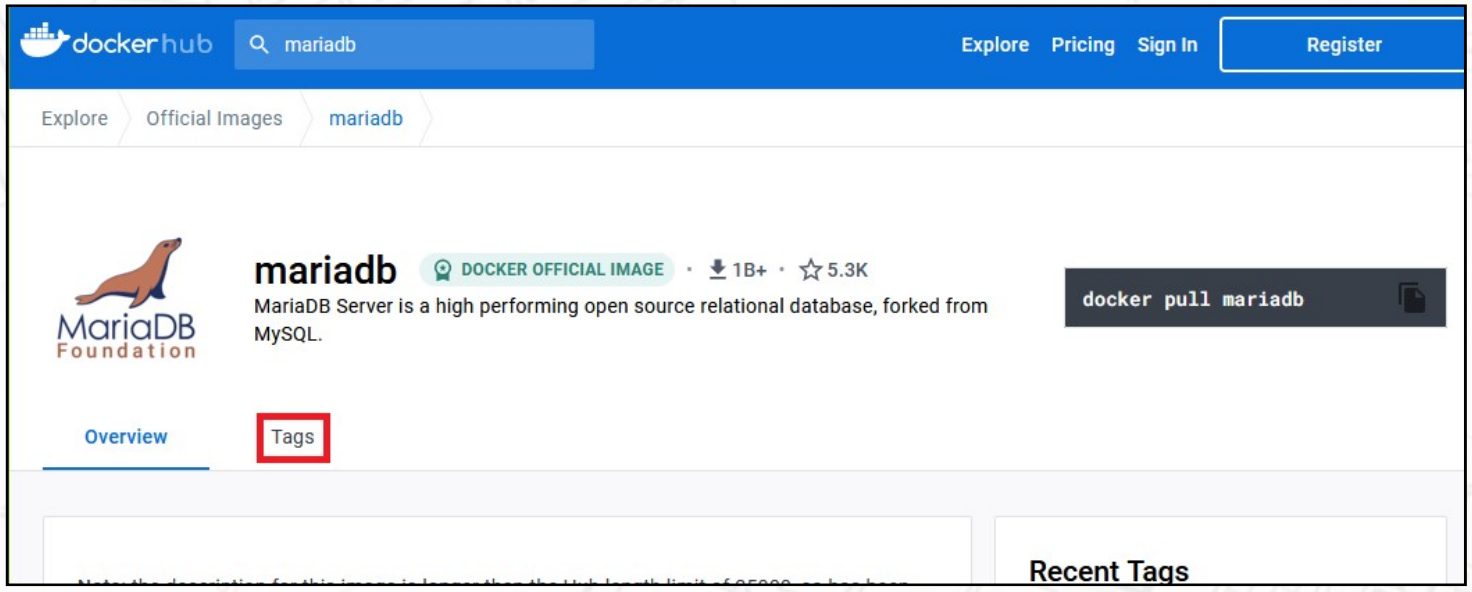
– Driver de conexión : lo determinará el conector:

conn = DriverManager.getConnection("jdbc:**mariadb**://localhost:3306/libreria", "root", "rootpwd");

conn = DriverManager.getConnection("jdbc:**mysql**://localhost:3306/libreria", "root", "rootpwd");

4 - SERVICIO EN DOCKER

- ✓ A) IMAGEN DOCKER MARIADB
  - https://hub.docker.com/_/mariadb
  - Nos vamos a TAGS (enlaces de descarga)



4 - SERVICIO EN DOCKER

✓ A) IMAGEN DOCKER MARIADB

– Descargar imagen:

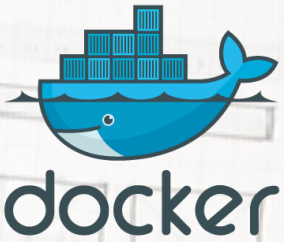
- sudo docker pull mariadb:latest

– Ejecutar imagen :

- sudo docker run -d --restart=always --name **servidorbbdd** --env MARIADB_USER=usuario --env MARIADB_PASSWORD=**usuariopwd** --env MARIADB_ROOT_PASSWORD=**rootpwd** -p 3306:3306 mariadb:latest

(En principio NO sería necesario exponer el puerto 3306 porque lo vamos a enlazar con otro contenedor que sí tendrá acceso a él. Pero como lo necesito para acceder desde JAVA, sí que se expone). restart=always hace que le lance al iniciar docker o se reinicie si hay algún problema

4 - SERVICIO EN DOCKER

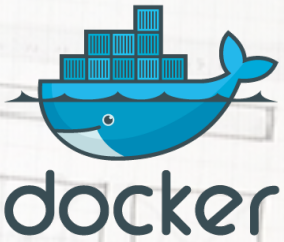


- ✓ A) IMAGEN DOCKER PHPMYADMIN
- – https://hub.docker.com/_/mariadb
- – Nos vamos a TAGS o etiquetas (enlaces de descarga)
- – Descargar imagen:
  - docker pull mariadb:latest
- – Ejecutar imagen:
  - sudo docker run -d --restart=always --name miphpma --link servidorbbdd:**db** -p 8080:80 phpmyadmin

(SÍ es necesario exponer el puerto 80 porque lo necesitamos para acceder desde fuera desde 8080)

db es el nombre interno del enlace que tendrá el servidor si se consultara desde servidorbbdd

4 - SERVICIO EN DOCKER



✓ B) IMAGEN DOCKER MYSQL

– https://hub.docker.com/r/mattrayner/lamp

– Nos vamos a TAGS o etiquetas (enlaces de descarga)

– Descargar imagen:

- docker pull mattrayner/lamp:latest-1804

– Ejecutar imagen (exponiendo además puerto 3306):

- sudo docker run --name LAMP --restart=always -p "80:80" **-p "3306:3306"** -v /home/usuario/app:/app mattrayner/lamp:latest-1804

4 - SERVICIO EN DOCKER

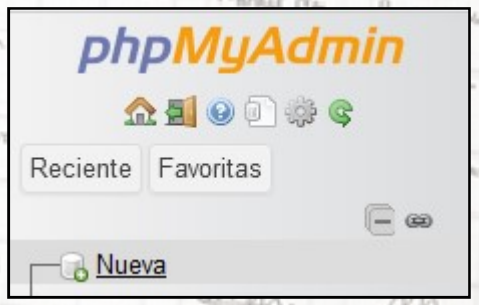
✓ PASOS DE INSTALACIÓN COMUNES (A y B)

– Abrimos phpMyAdmin en el puerto http://localhost:<puerto>/phpmyadmin utilizando user y password administrador al crear el contenedor (dado como parámetro o devuelvo por el contenedor)

– Se crea la base de datos en el menú superior (click en Nueva). Se le pone un nombre (libreria)

– Hacemos click en Importar

– Elegimos fichero *libreria.sql*

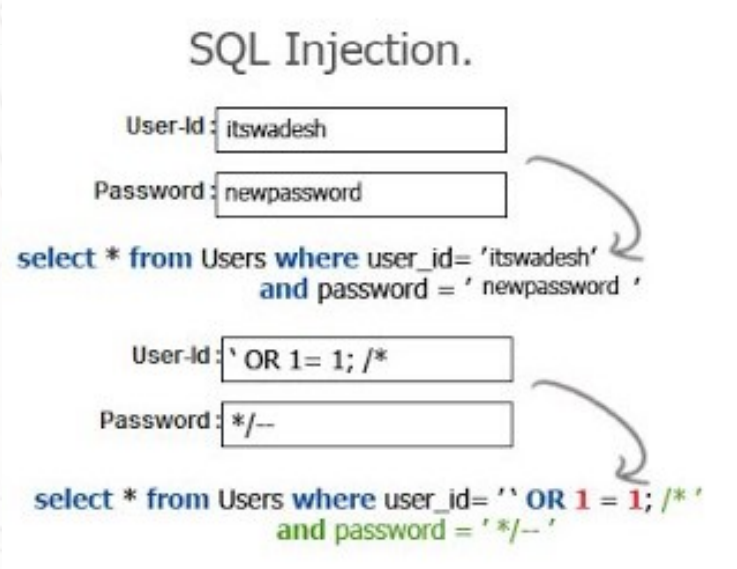


5 - CRUD

- ✓ CRUD: **C**reate **R**etrieve **U**pdate **D**elete
- ✓ **Conexión JDBC** (Java DataBase Connectivity - 1ºDAM): donde se indica: tipo, ip:puerto, user y pwd. Usa un statement (declaración) para ejecutar consultas.
  - **con**=DriverManager.getConnection("**jdbc**:**mariadb**://localhost:3306/libreria", "root", "rootpwd");
- ✓ **SELECT:**
  - **con**.createStatement.executeQuery(sql)
  - Devuelve ResultSet → las metemos en un OL, que a su vez va asociado a un TableView
- ✓ **INSERT, UPDATE, DELETE:**
  - **con**.createStatement.executeUpdate(sql)

5 - CRUD

- ✓ **INYECCIÓN SQL**: Problema que afecta a cualquier SQL generada sin seguridad.
- ✓ Afecta a cualquier plataforma donde se pueda “colar” código (web, apps, escritorio) vía SQL.

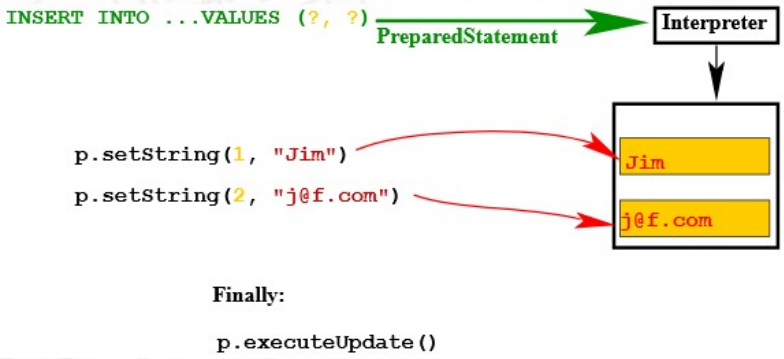


-La primera comilla cierra el campo de id
-el OR 1=1 siempre será TRUE
-el /* es un comentario */
Y el -- indica que lo que venga de SQL será ignorado

CONCLUSIÓN: Se accede a todos los datos de la tabla

5 - CRUD

- ✓ **ANTIINYECCIÓN SQL**:
- ✓ Asocia “?” con posición y tipo de dato en tiempo de ejecución, reduciendo así el peligro de la antiinyección. Ej:



- ✓ Tipos de datos: .setString, .setInt, .setDouble, .setDate, .setFloat, .setBoolean, .setBool, etc…

5 - CRUD

✓ CRUD: **C**reate **R**etrieve **U**pdate **D**elete

– Vemos en el proyecto de LIBROS un ejemplo de CRUD con JAVAFX y MariaDB . **[crud_javafx_libros_base]**

• Vemos tanto la cadena de conexión, como la antiinyección y la asociación con un TableView.

• OJO!! Es una versión básica con !!!errores de Usabilidad Y Funcionalidad!!!. Saltan excepciones si no se hace como está previsto.

– Posteriormente veremos otros ejemplos donde habrá:

• Errores arreglados (validación de campos)

• CSS, animaciones, popup menú, adornos, etc…