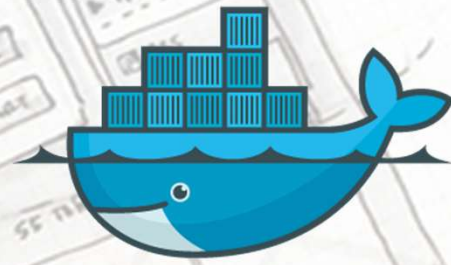


DESARROLLO DE INTERFACES

2º DAM

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



docker



TEMA 2-3 – ORGANIZACIÓN VENTANAS Y
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).

```
git clone https://github.com/MolinaJM/DI-T2-3.git
```

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

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.. Son: `.getScene()` y `.getWindow()` y se pueden aplicar a CUALQUIER NODO!. Ej: `cajatexto`.
- **Scene** `miScene = (Scene) this.cajatexto.getScene();`
- **Stage** `miStage = (Stage) this.cajatexto.getScene().getWindow();`
- **Usos:** añadir un nodo por código al scene o para ver atributos de ventana (stage), para acceder a etiquetas FXML, asignar nuevos eventos, etc etc.
- **Excepción:** se puede hacer desde cualquier sitio EXCEPTO en **Initialize: el Scene y Stage aún no están disponibles (porque aún no se ha mostrado nada y por tanto → nada es accesible**. Si queremos hacer esto tendremos que usar un HILO! (ver ejemplo de paso de variable **v2**)
- Curiosamente es en el initialize donde más nos hará falta acceder a los componentes.

1 – ALTERNATIVAS DE DISEÑO



- ✓ Ejemplos de Línea que da error y forma correcta de consultar:

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    Stage stageC = (Stage) this.miEtiqueta.getScene().getWindow();
    //No se puede acceder a los eventos de SCENE O STAGE en Initialize a no ser que se use un hilo
    Platform.runLater(() -> {
        //Accedemos al stage actual mediante cualquier nodo
        Stage stageB = (Stage) this.miEtiqueta.getScene().getWindow();
        stageB.setOnCloseRequest(e -> {
            e.consume();//Detiene el evento de cierre
            stageB.hide();//Oculta ventana
        });
    });
}
```

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 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. **Esto es MUY ÚTIL ya que estamos “cogiendo” un nodo de otra ventana para hacer algo con él!!:**
 - ✓ Es necesario tener acceso al scene que lo contiene.
 - ✓ Ejemplo en v3:
 - `TextField t2 = (TextField) scene.lookup("#cajaTexto2");//Acceso por ID`

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.1) 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...
- **A.2) USANDO UNA ÚNICA VENTANA(SCENE) EN 1 FXML pero cargando PARTES (FXMLs) a posteriori:** Esta variante se ve después de la B, ya que, una vez entendido el paso de controladores, se entiende mejor esta variante.

1 – ALTERNATIVAS DE DISEÑO



- **B) USANDO DISTINTAS STAGES (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???**

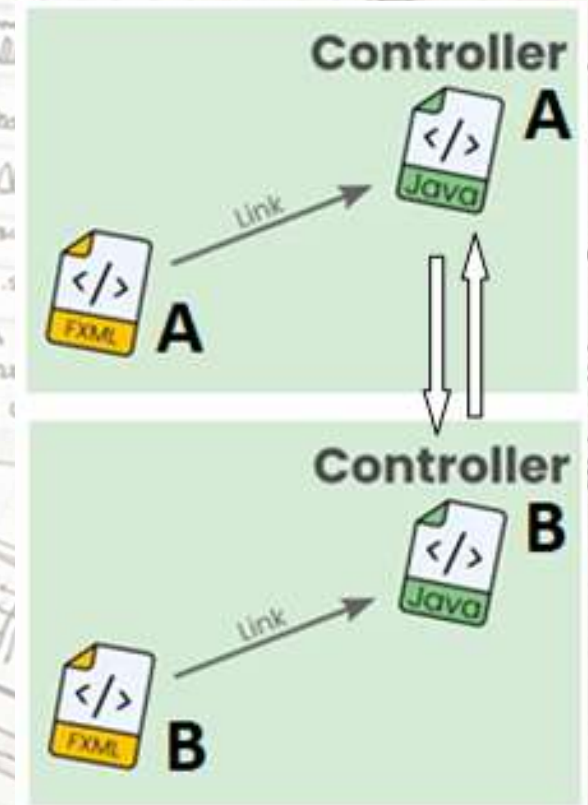
1 – ALTERNATIVAS DE DISEÑO



- ✓ **PASO DE VARIABLES:** Hay distintas formas de hacer esto, cada una con sus ventajas e inconvenientes. Vemos algunas formas, no son ni las mejores ni las únicas, se puede hacer de más formas:

[\[javafx-ejemplos-pasovariables\]](#)

- ✓ **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 principio de encapsulado)



1 – ALTERNATIVAS DE DISEÑO

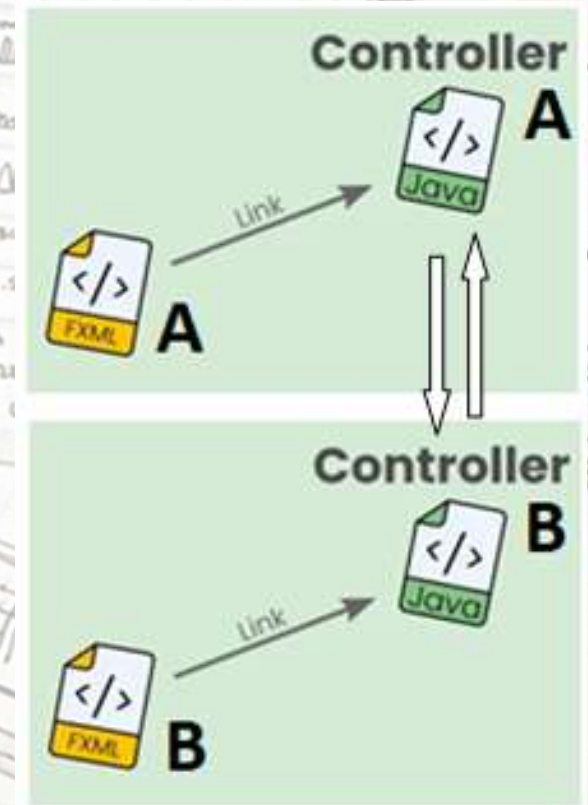


✓ ~~Comunicar controladores [v1]~~

- ~~Esta versión no se ve porque es una variante ineficiente del paso de controladores, la cual se optimiza en la v2.~~

– Comunicar controladores [v2]

- Mecanismo estándar de comunicación de objetos (pasando el controlador mediante un setter).



1 – ALTERNATIVAS DE DISEÑO



- ✓ 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.
- ✓ Necesitan la creación de un modelo (dato que se lee/escribe). Ej: modelo Etiqueta(String,Integer) lo ancho al Stage.

```
stageB.setUserData(e1); //Setter
```

```
Etiqueta e2 = (Etiqueta) stageB.getUserData();//Gettter
```

- **Inconveniente:**

- hace el código un poco más enrevesado.

- **Ventajas:**

- No hay que hacer métodos públicos
- Muy **versátil**

Método de las “etiquetas”



1 – ALTERNATIVAS DE DISEÑO



- ✓ **Mediante clase Singleton [v4].** Una clase *Singleton* garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia. Es decir, evita llamadas concurrentes.
- ✓ Dicho de otra forma: lo usamos como “almacén” global para variables que podemos guardar/consultar. Y esa variable puede ser de **CUALQUIER TIPO (String, Integer, Modelo, Clase, OL, etc..)**. Ej: Clase Conector de BBDD.
- ✓ Se crea un método “constructor” el cuál es *synchronized* para asegurarnos que solo un hilo puede ejecutar este método a la vez (asegura inconsistencias al trabajar con recursos compartidos o al usar concurrencia)

1 – ALTERNATIVAS DE DISEÑO



- **Mediante clase Singleton [v4].** Una clase *Singleton* garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia. Es decir, evita llamadas concurrentes.
- **Inconveniente:**
 - Si no se implementa bien podemos tener problemas de concurrencia.
- **Ventajas:**
 - Crea un punto único para recursos compartidos (una especie de almacén de variables global)
 - Consecuencia de lo anterior es que optimizamos código, al tener un único sitio donde guardar variables evitamos crearlas múltiples veces (por ejemplo, así no repetir conexiones a la base de datos).

1 – ALTERNATIVAS DE DISEÑO



- **A.2) USANDO UNA ÚNICA VENTANA(SCENE) EN 1 FXML** pero cargando **PARTES** a posteriori : Es el mismo caso anterior, pero lo que tenemos en el FXML principal es el “esqueleto” de la APP. El contenido de cada una de las partes de dicha ventana podría cargarse desde distintos FXML. Además, habría que “comunicar” los controladores.
- **Ventajas:** no tenemos todo el código en una única clase
- **Inconvenientes:** necesita paso de variables (para comunicar los Controladores)

1 – ALTERNATIVAS DE DISEÑO



- **A.2) USANDO UNA ÚNICA VENTANA(SCENE) EN 1 FXML pero cargando PARTES a posterior:** El código sería algo así (se ha resumido):

```
public void start(Stage primaryStage) throws Exception {  
    //Cargamos el archivo Main.fxml principal  
    FXMLLoader mainLoader = new FXMLLoader(getClass().getResource("Main.fxml"));  
    Parent mainRoot = mainLoader.load();  
    MainController mainController = mainLoader.getController();  
  
    // Cargar el archivo LeftView.fxml y RightView.fxml  
    FXMLLoader leftLoader = new FXMLLoader(getClass().getResource("LeftView.fxml"));  
    Parent leftView = leftLoader.load();  
    FXMLLoader rightLoader = new FXMLLoader(getClass().getResource("RightView.fxml"));  
    Parent rightView = rightLoader.load();  
    //Obtenemos controladores  
    LeftController leftController = leftLoader.getController(); //Desde aquí puedo llamar a las funciones de left!!!  
    RightController rightController = rightLoader.getController(); // Desde aquí puedo llamar a las funciones de right!!!  
  
    //Añadimos las vistas cargadas al HBox del controlador principal  
    Parent root = mainController.getRoot();  
    if (root instanceof HBox) {  
        HBox hBox = (HBox) root;  
        hBox.getChildren().addAll(leftView, rightView);  
    }  
}
```


1 – ALTERNATIVAS DE DISEÑO



■ A.2) USANDO UNA ÚNICA VENTANA(SCENE) EN 1 FXML pero cargando PARTES a posterior : (SIGUE)

//Ahora, ¿cómo comunicamos algo desde el subcontrolador al principal, al mainController? Debemos pasarle una referencia con un setter.

//En el mainController:

```
leftController.setMainController(mainController);  
rightController.setMainController(mainController);
```

//Ejemplo en LeftController.java

```
private MainController mainController;
```

```
public void setMainController(MainController mainController) {  
    this.mainController = mainController;  
}
```

```
public void onButtonClick() {  
    mainController.notifyRight("Mensaje desde la izquierda");  
    mainController.envíaDatosADerecha ("HOLA"); //Hipotética función de comunicación de datos  
}
```

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** (ej: php, springboot, etc..)

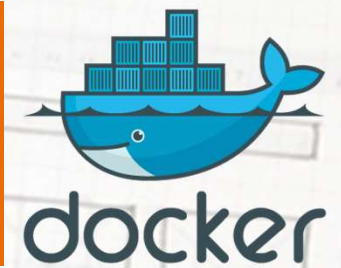
3 – MARIADB VS MYSQL



MARIADB VS MYSQL

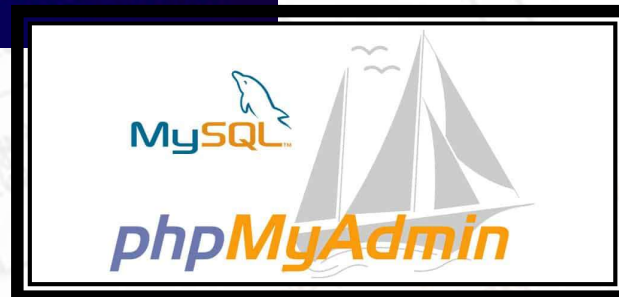
- ✓ Aunque MariaDB es una bifurcación de MySQL, estos dos sistemas de gestión de bases de datos siguen siendo bastante diferentes:
 - MariaDB tiene licencia GPL mientras que MySQL tiene un enfoque de doble licencia (desde que lo compró ORACLE). No es libre (no accedes a todas las características) pero 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

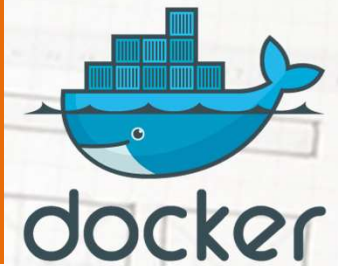


✓ MONTANDO SERVICIOS EN DOCKER

– Ya sabemos trabajar con Docker, ahora solo falta instalar los contenedores que nos hagan falta. 2 Alternativas:



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");
```

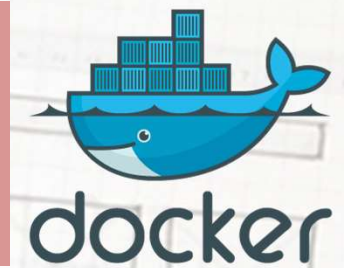
- **B) Contenedor MySQL + enlace a Phpmyadmin.**

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

- Driver de conexión: conn =

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/libreria", "admin",  
"jmb6M9xC4OPR")
```

4 - SERVICIO EN DOCKER



✓ MONTANDO SERVICIOS EN DOCKER

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

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

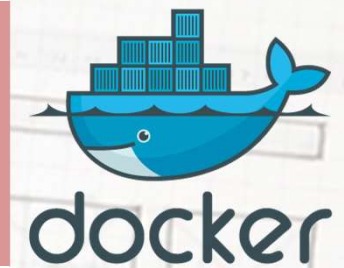
– Driver de conexión : lo determinará el conector:

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

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

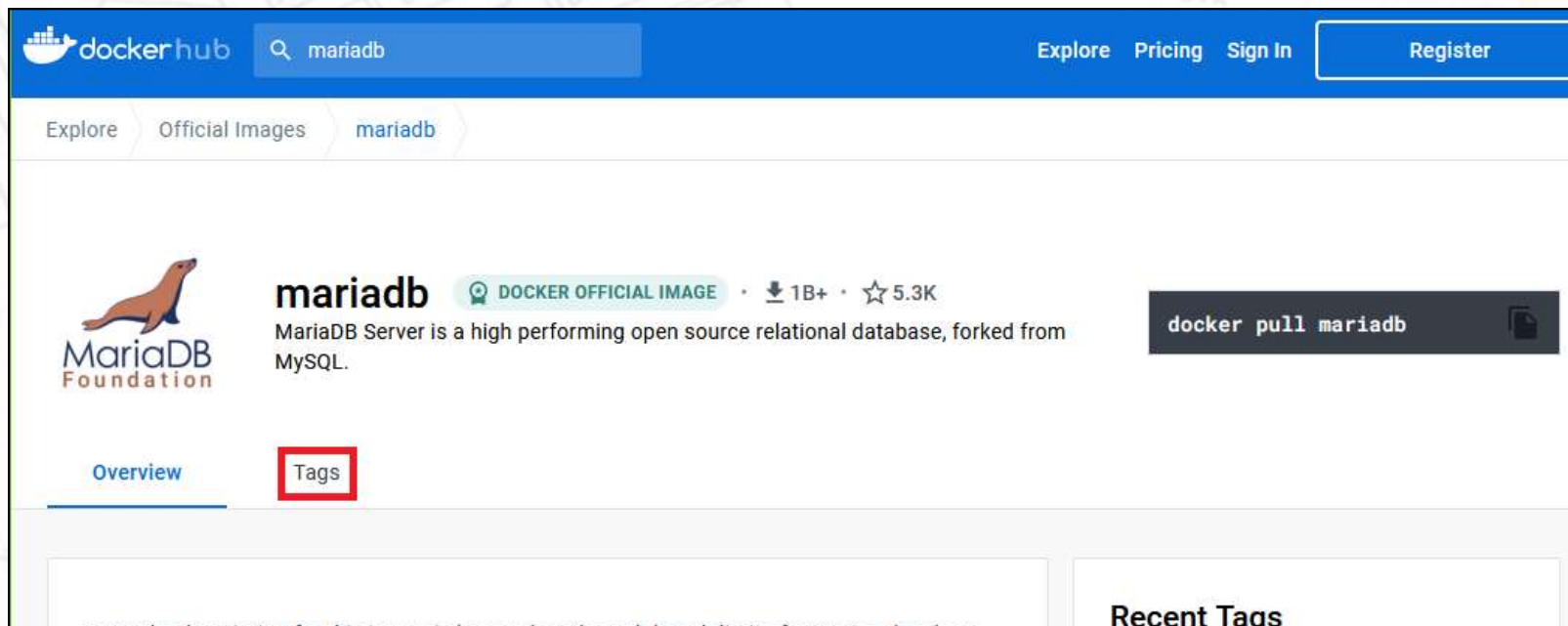
**PODEMOS USAR LOS DOS INDISTINTAMENTE!!!
AUNQUE ES RECOMENDABLE USAR MYSQL**

4 - SERVICIO EN DOCKER

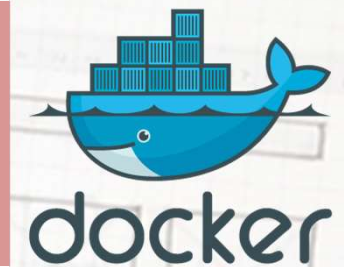


✓ A) IMAGEN DOCKER MARIADB

- https://hub.docker.com/_/mariadb
- Nos vamos a TAGS (enlaces de descarga)



4 - SERVICIO EN DOCKER



✓ A) IMAGEN DOCKER MARIADB

– Descargar imagen:

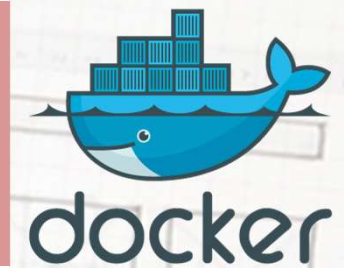
- `sudo docker pull mariadb:latest`

– 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`

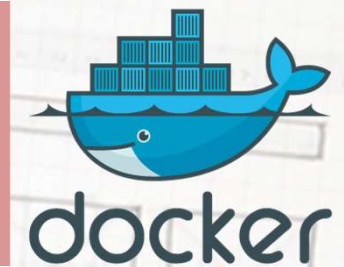
– Ejecutar imagen:

- `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)

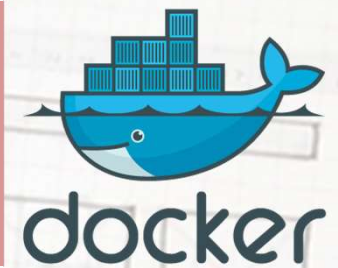
db es el nombre interno del enlace que tendrá el servidor si se consultara desde servidorbbdd

4 - SERVICIO EN DOCKER



- ✓ B) IMAGEN DOCKER LAMP (**USAMOS ESTA!**)
- <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: Create Retrieve Update Delete**

✓ **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"); \\ Similar para MYSQL

✓ **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.

SQL Injection.

User-Id:
Password:

`select * from Users where user_id= 'itswadesh'
and password = ' newpassword '`

User-Id:
Password:

`select * from Users where user_id= '' OR 1 = 1; /* '
and password = ' */-- '`

-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

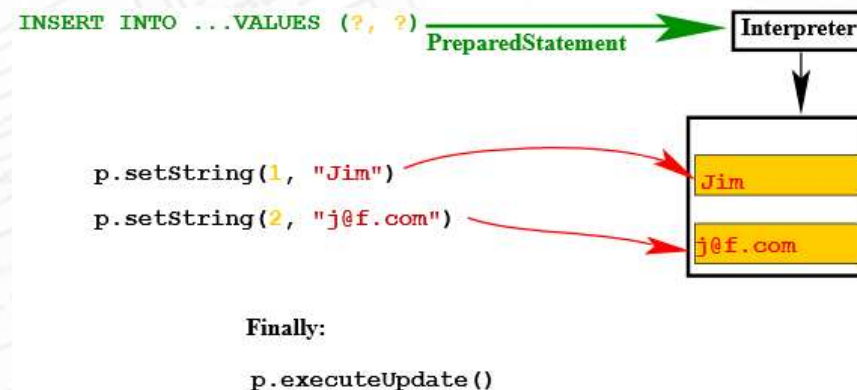
CONCLUSIÓN: Se accede a todos los datos de la tabla

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**ppdate **D**elte

– 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...