ACCESO A DATOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

BBDD embebidas e independientes

Gestores de bases de datos embebidos e independientes

Diferentes **variantes, opciones o posibilidades** **que tenemos de almacenamiento** de datos en sistemas de bases de datos relacionales:

• **Bases de datos en memoria**: Almacena toda la información de la misma en **memoria principal del sistema (RAM)**. **No** se almacenará ningún dato **en disco**. Es muy rápida ya que el acceso a la memoria RAM es mucho más rápido que que el acceso a disco.

• **Bases de datos embebidas**: Es **parte de la aplicación** que se ha desarrollado. Accederemos por medio de los **JDBC driver**. El **motor de la base de datos** corre con el **mismo motor de la aplicación** Java (JVM-Java Virtual Machine) mientras la aplicación está en ejecución. **Desventaja:** la **cohesión** y la **dificultad de mantenimiento** que podemos encontrar en estos casos. Encaja perfectamente con la idea de tener un **repositorio** para **persistir las transacciones** sin ningún tipo de intervención por parte del usuario.

• **Bases de datos independientes SGBD** (Sistemas de Gestión de Bases de Datos relacionales): Tenemos un **gestor o administrador** de base de datos dedicado a **resolver** ciertos problemas de **mantenimiento**. Las bases de datos **cliente/servidor** son **independientes** y las más **pesadas y potentes** (cantidad de **procesos adicionales**).

Gestores de base de datos embebidos

• **HyperSQL**: se ajusta a la versión estándar **SQL 2011** y a la especificación **JDBC 4**. Además, **soporta** cada característica clásica de las **bases de datos modernas** de tipo **relacionales**. Se puede ejecutar tanto en modo **embebido** como en modo cliente/**servidor**. La base de datos es bastante **estable**, es de una **confianza** considerable, como los proyectos desarrollados de código abierto "OpenOffice" y "LibreOffice".

Desarrollado en **Java**, corre sin problema en nuestra **JVM**, facilitándonos una **interfaz JDBC** para el **acceso** a datos. El paquete principal que nos descargamos contiene un fichero .**jar** llamado "**hsqldb.jar**" dentro del directorio **/lib**, que contiene el componente requerido: el motor de la **base de datos HyperSQL RDBMS** y el **driver JDBC** embebido en la aplicación Java.

A partir de la **versión 2.3.X** en adelante soporta el mecanismo **MVCC (Multi version concurrency control)**:

> Este concepto sienta su base cuando **dos o más usuarios** de bases de datos **acceden** a la misma. Con el concepto MVCC **evitamos el tema de los bloqueos**, ya que lo que se estaría **viendo** con nuestro **acceso** es algo parecido a una **imagen** de la base de datos, por tanto, **no interferiríamos** en las transacciones.

• **ObjectDB**: también contiene los dos modos, **embebido** y modo cliente/**servidor**. No es exactamente una base de datos relacional, sino una **base de datos orientada a objetos** con soporte para la especificación **JPA2 (Java Persistence API)**. Como resultado, el uso de una **capa de abstracción**, como la de **Hibernate**, tiene un **mejor rendimiento** que cualquier otra base de datos. Debido a su compatibilidad por defecto con JPA, **se podría eliminar la capa ORM** directamente, si es que nuestro aplicativo tuviera una para aprovechar el rendimiento. Un ORM (**Object Relational Mapping**) es un **modelo de programación** cuya misión es **transformar las tablas de una base de datos** de forma que las tareas básicas, que realizan los programadores, estén simplificadas.

• **Java DB y Apache Derby**: Son muy similares, de hecho, **Java DB** está construida **sobre el motor** de la base de datos **Derby**. Esta está escrita por completo en el lenguaje **Java** y puede ser incluida, **fácilmente**, en cualquier **aplicación Java**.

Derby **soporta** todas las funcionalidades estándar de una base de datos **relacional**. Puede ser desplegada en el modo simple **embebido** o también en el modo cliente/**servidor** como sus competidores. Para **embeber** nuestra base de datos Derby en nuestra aplicación Java, simplemente, hay que incluir en nuestro proyecto Java el fichero "**derby.jar**" del directorio **/lib**. Este fichero contiene tanto el propio **motor** de la base de datos como los **conectores** necesarios para realizar la **conexión con el driver JDBC** desde código.

- **H2 Database**: las principales características de la base de datos H2 son:
    - Es muy **rápida**, de **código abierto**, **JDBC** API.
    - Contiene modo cliente/**servidor**, modo **embebido** y modo **desplegable en memoria**.
    - Se puede usar perfectamente para **aplicaciones web**.
    - Muy manejable y **transportable**, el fichero .jar ocupa **2MB** de espacio total.
    - Soporta **MVCC** (Multiversion concurrency control).
    - Se puede usar **Postgress OBDC** driver. PostgreSQL es un **sistema o motor de bases de datos** compatible con los **servicios de OVHcloud** y la mayoría de las herramientas más populares del mercado. Es compatible con diversos **modelos de datos para crear aplicaciones orientadas a objetos**, potentes y escalables.

"Spring boot database" es de ayuda para crear proyectos (incluidos los web) con SGBD embebidos.

Comparativa de gestores de bases de datos embebidos

	H2	DERBY	HYPERSQL	MYSQL	POSTGRESS
Java Puro	SI	SI	SI	NO	NO
Modo memoria	SI	SI	SI	NO	NO
Base de datos encriptada	SI	SI	SI	NO	NO
Driver ODBC	SI	NO	NO	SI	SI
Búsqueda texto completo	SI	NO	NO	SI	SI
MVCC	SI		+versión 2.3.X	SI	SI
Espacio (embebido)	~2 MB	~3 MB	~1.5 MB		
Espacio (cliente)	~500 KB	~600 KB	~1.5 MB	~1 MB	~700 KB

H2: buena opción para agregar una base de datos **embebida**, en memoria, o de tipo cliente/servidor.

Instalación de base de datos H2 en aplicación web Java usando Spring Boot

Spring es un framework muy conocido. Spring Boot es un **módulo dentro de Spring:** hace **fácil** ciertas partes del proceso que por defecto son más complejas.

En un proyecto **web Java** con **Maven** existen varias etapas como seleccionar las **dependencias** de las librerías que vamos a usar, construir el aplicativo o desplegarlo a un servidor. Spring Boot facilita al máximo la **selección de dependencias** el **despliegue de la aplicación**.

Nos será de gran utilidad añadir una base de datos que facilite el almacenamiento de información para nuestra aplicación web.

En https://start.spring.io/ se nos ofrece un entorno sencillo para crear el arquetipo de nuestra aplicación con distintas variables:

• **Opción project**, elegiremos **maven**. Ya que nuestro ejemplo será implementado y compilado con **dependencias maven**.

• **Opción Spring boot**, dejaremos la release version que venga **por defecto**, ya que será **la más estable** por el momento.

• **Opción project metadata**:

- **Group**: podemos dejarlo tal como está, ya que vamos a realizar una **prueba**.
- **Artifact**: igual que el anterior.
- **Name**: la misma idea, podemos dejarlo como aplicación **Demo**.
- **Description**: incluiremos una descripción, por ejemplo, “Hola mundo con base de datos embebida H2”.
- **Package name**: podemos dejar el nombre que nos asigna **automáticamente**.
- **Empaquetado**: elegiremos .**jar**.
- **JDK Java**: podemos trabajar bien con el **JDK 8**.

• En **Dependencies** incluiremos nuestra **base de datos H2 embebida**.

Nos **permite incluir la mayoría de los framework y utilidades** que rodean el lenguaje por medio de **dependencias** que serán incluidas **automáticamente**.

Incluiremos nuestra base de datos,

Hemos **añadido la base de datos H2** y el módulo "**Spring Web**" (vamos a simular la funcionalidad de una **aplicación web** cliente/servidor). Tambien incluiremos **Spring Data JPA**. Hacemos clic en "GENERATE" y automáticamente se nos descargará un fichero con extensión .zip o .rar, y tendremos un proyecto web java con todas las características.

Descomprimimos el directorio del proyecto, y lo abrimos **desde el IDE** que usemos frecuentemente (Intellij, Eclipse, Netbeans...). En el archivo "**pom.xml**" Spring Boot nos ha añadido por defecto algunas de las **dependencias** que nosotros le indicamos anteriormente en la interfaz:

```
<dependency>
	<groupId>com.h2database</groupId>
	<artifactId>h2</artifactId>
	<scope>runtime</scope>
</dependency>
```

Las **librerías** de la **base de datos H2** han sido **añadidas** y perfectamente **integradas** por Spring Boot.

Nos genera un **fichero** llamado "**Application.properties**" (en **src/main/resources**/ de la raiz del proyecto) donde tenemos algunos aspectos de **configuración básica** de nuestra **base de datos H2**:

Lo siguiente no es necesario agragarlo al archivo para que funcione nuestra aplicación web, aunque debemos de entender qué hace cada configuración:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Con la instrucción `spring.datasource.url=jdbc:h2:mem:testdb`, podemos ver como Spring Boot nos ha configurado nuestra base de datos para que el almacenamiento de la misma se realice en **memoria del sistema**. Evidentemente, como es volátil, la información estará visible mientras la aplicación esté corriendo, pero no hay ningún tipo de problema, porque si queremos cambiar a tipo **embebida con almacenamiento en disco** ejecutaremos algo así:

```
spring.datasource.url=jdbc:h2:file:~/data/demo
```

Con esto se guardaría en disco los cambios en nuestra base de datos. Nuestra **base de datos quedará almacenada en la ruta** que se indica.

Con **spring.datasource.driverClassName** indicaremos el nombre del driver en el fichero .properties de Spring Boot, y a continuación, deberíamos indicar el nombre relativo del driver.

Con **spring.datasource.username=sa** y **spring.datasource.password=password** agregamos las credenciales para la conexión a la base de datos. Estas las podremos cambiar desde este archivos cuando lo necesitemos y surtirán efecto en ese momento.

Iniciar la BBDD

Nuestra base de datos está montada y preparada para funcionar. Sería ideal montar una buena **capa DAO** (patrón **Data Access Object**) de acceso a la capa de datos en nuestro aplicativo, ya que tendremos todas las opciones de persistencia y el potencial que nos ofrece **JPA** (JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite **interactuar con la base de datos por medio de objetos**). Nosotros, para hacer **un test** de la misma, vamos a indicar otros ficheros de configuración.

En la ruta "**src/main/resources**" añadiremos un fichero llamado "**data.sql**". La próxima vez que arranque nuestra aplicación, **Spring Boot** cogerá dicho fichero y **lo ejecutará** al comienzo de la misma, por lo tanto, este será un buen momento para la **creación de una tabla**, o borrado de alguna que estuviera con anterioridad; en definitiva, realizar **operaciones de limpieza y preparado de datos**:

En principio para que nuestra aplicación funcione tampoco es necesario agregar este archivo y añadir estas sentencias SQL, pero es conveniente saber utilizarlo:

```
DROP TABLE IF EXISTS RESERVATION;
CREATE TABLE RESERVATION (
    id INT AUTO_INCREMENT PRIMARY KEY,
    reservation_date TIMESTAMP NULL DEFAULT NULL,
    name VARCHAR(250) NULL,
    status VARCHAR(25) NOT NULL DEFAULT 'FREE'
    );
```

Otro dato **importante** a tener en cuenta es cómo accedemos a la **consola de nuestra base de datos** o a nuestro panel de gestión de la base de datos. Añadimos en nuestro fichero "**application.properties**" la siguiente línea:

```
spring.h2.console.enabled=true
```

Cuando nuestra aplicación esté ejecutándose, nos dirigiremos a nuestro navegador e introduciremos la siguiente URL:

http://localhost:8080/h2-console

De esta forma, accederemos a nuestro **panel de control de la base de datos**, donde podremos realizar distintas operaciones.

Dos maneras de agregar **base de datos Apache Derby** a nuestra aplicación de manera **embebida**:

1. Descargar el .**jar** de la página oficial de Apache Derby:

https://db.apache.org/derby/derby_downloads. html#For+Java+8+and+Higher.

Y añadiremos este fichero a nuestro aplicativo, el cual contiene el motor de la base de datos y el conector.

2. Directamente montar nuestro proyecto con **Spring Boot:** En spring.io elegiremos la **dependencia de Derby DB (Apache Derby Database sql)** para la creación de nuestro proyecto atendido por Spring Boot.

Gestores de base de datos independientes

Una base de datos independiente no puede ejecutarse bajo la misma máquina virtual de Java que usa nuestra aplicación en ejecución. Puede ser instalada **en local**, pero **con fines de prueba o aprendizaje** en la mayoría de casos. **Lo común** es tener una **base de datos independiente** separada e instalada **en otra máquina**.

Consumirá siempre más recursos que una embebida. Al estar en una **máquina aislada** para dicha base de datos, **aprovechará mejor los recursos** que tiene disponibles. Se preparan en dicha máquina separada y aislada para **volcar todo el potencial** que ofrece con todos sus **procedimientos y operativos**.

Las **más conocidas** a nivel comercial y de mayor potencia y extensión:

- **SQL Server DataBase.**
- **Oracle.**
- **Postgres SQL.**
- **MySql.**