MEDAC
Instituto Oficial de Formación Profesional

ACCESO A DATOS
TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Manejo de conectores II

05

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Gestores de bases de datos embebidos e independientes 4

/ 3. Gestores de base de datos embebidos I: HyperSQL y ObjectDB 4

/ 4. Gestores de base de datos embebidos II: Java DB, Apache Derby y H2 5

/ 5. Caso práctico 1: "Embebida o independiente" 6

/ 6. Gestores de bases de datos embebidos III: Comparativa 6

/ 7. Base de datos embebida en aplicación web a través de Spring Boot I: Añadir BBDD 7

/ 8. Base de datos embebida en aplicación web a través de Spring boot II: IDE 8

/ 9. Base de datos embebida en aplicación web a través de Spring Boot III: Inicio BBDD 9

/ 10. Gestores de base de datos independientes 10

/ 11. Caso práctico 2: "Cambio de credenciales" 11

/ 12. Resumen y resolución del caso práctico de la unidad 12

/ 13. Webgrafía 13

OBJETIVOS

Conocer las bases de datos embebidas más importantes del mercado.

Conocer las bases de datos independientes más importantes del mercado.

Saber escoger entre unas y otras según la línea de negocio o potencial de nuestra aplicación.

/ 1. Introducción y contextualización práctica

En la siguiente unidad didáctica, estudiaremos los distintos sistemas de gestión de datos. Realizaremos un estudio de qué sistema podremos incluir según las características de nuestra aplicación.

Profundizaremos en las bases de tipo embebida en todas sus modalidades y veremos una comparativa de las distintas opciones que tenemos en el mercado. Elegiremos una y realizaremos su instalación.

Analizaremos las diferencias entre bases de datos embebidas e independientes, y mostraremos las más comunes independientes.

Fig. 1. Conectores..

Planteamiento del caso práctico inicial

A continuación, vamos a plantear un caso a través del cual podremos aproximarnos de forma práctica a la teoría de este tema.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.

Audio intro. "Inserta base de datos embebida"
https://bit.ly/2Wkl9J6

/ 2. Gestores de bases de datos embebidos e independientes

En esta unidad, hablaremos de las diferentes **variantes que tenemos de almacenamiento de datos en sistemas de bases de datos relacionales.** Variantes que siempre giran alrededor del desarrollo de nuestro aplicativo. Realmente, tenemos varias opciones o posibilidades:

- **Bases de datos en memoria:** es importante que no confundamos el término de base de datos cargada en memoria y base de datos embebida. Una base de datos en memoria, **almacena toda la información de la misma en memoria principal del sistema** (RAM). Como bien sabemos, esta memoria es volátil y la información solo permanece mientras la aplicación esté ejecutándose. Por lo tanto, no se almacenará ningún dato en disco.

- **Bases de datos embebidas:** es aquella que **es parte de la aplicación que se ha desarrollado.** Accederemos a este tipo de bases de datos por medio de los ya conocidos 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. Una posible desventaja de usar este tipo de base de datos puede ser la cohesión y la dificultad de mantenimiento que podemos encontrar en estos casos. De hecho, este tipo de base de datos no es construida para lidiar con muchos problemas. 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:** por el contrario, aquí sí **tenemos un gestor o administrador de base de datos dedicado a resolver ciertos problemas de mantenimiento.** Son los **SGBD**, Sistemas de Gestión de Bases de Datos relacionales. Las bases de datos cliente/servidor son independientes y, obviamente, las más pesadas y potentes. Nos referimos a pesadas no por lentitud, sino por la cantidad de procesos adicionales que engloba una base de datos de este tipo.

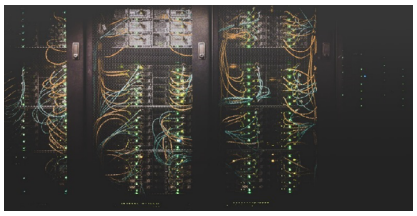

Fig. 2. Sistema base de datos independiente.

/ 3. Gestores de base de datos embebidos I: HyperSQL y ObjectDB

A continuación, realizaremos un estudio sobre los **sistemas gestores de base de datos embebidos del mercado según su usabilidad, características e importancia.** Disponemos de un amplio abanico para elegir con sus pros y sus contras. Los vemos a continuación:

- **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 servidor. La base de datos es bastante estable, es de una confianza considerable, como los proyectos desarrollados de código abierto "Open Office" y "LibreOffice". A partir de la versión 2.3.X en adelante soporta el mecanismo MVCC *(Multi version concurrency control)*.

Audio 1. "Multi version concurrency control"
https://bit.ly/30cYKyH

Como está **desarrollado en Java**, corre sin problema en nuestra **JVM** como comentamos anteriormente, 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.

Fig. 3. Logo de HyperSQL.

- **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. 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.

Fig. 4. Logo de ObjectDB.

/ 4. Gestores de base de datos embebidos II: Java DB, Apache Derby y H2

Otros de los sistemas gestores de base de datos embebidos del mercado pueden ser:

- **Java DB y Apache Derby:** estas dos bases de datos son muy similares, de hecho, J**ava 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.

Fig. 5. JavaDB y Apache Derby

- **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*.

Fig.6. H2 Database.

Vídeo 1. "Spring boot database"
https://bit.ly/392FJTM

/ 5. Caso práctico 1: "Embebida o independiente"

Planteamiento: A José le plantean el desarrollo de la capa de acceso a datos de una aplicación pensada para coger citas y almacenarlas en base de datos para un consultorio.

No serán más de diez filas al día en dos o tres tablas.

Nudo: ¿Qué tipo de base de datos tendrá que escoger?

Tras investigar, puede ver que existen bases de datos embebidas en la aplicación y también bases de datos independientes...

Desenlace: Debido a las especificaciones que se plantean, nuestro compañero, claramente, debería montar una base de datos embebida en la aplicación desarrollada.

Con ella, podrá realizar una gestión de casi todas las funcionalidades que brinda el lenguaje SQL, dado que la envergadura del planteamiento es simple y parece no necesitar desarrollar procedimientos costosos de base de datos, ni tampoco operaciones que consuman muchos recursos, por lo que no sería necesario montar una máquina a parte e instalar una base de datos independiente.

Definitivamente con una base de datos embebida corriendo sobre la misma JVM de nuestra aplicación sería suficiente.

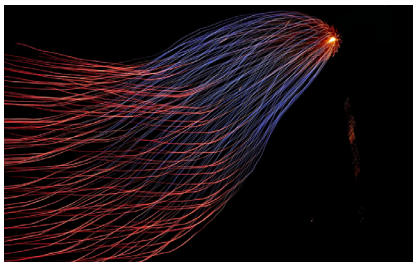

Fig. 7. Embebida o independiente.

/ 6. Gestores de bases de datos embebidos III: Comparativa

Una vez vistos los sistemas de datos embebidos más importantes del mercado, realizaremos una comparativa con algunas de las bases de datos presentadas, de tal forma que nos sea más fácil su elección.

EJEMPLO	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			SI	SI
Espacio (embebido)	~2 MB	~3 MB	~1.5 MB	-	-
Espacio (cliente)	~500 KB	~600 KB	~1.5 MB	~1 MB	~700 KB

Tabla 1. Comparativa diferentes sistemas bases de datos.

En la tabla anterior, podemos ver las distintas bases de datos enfrentadas por diferentes características. Algunas de las características son: el lenguaje que están desarrollado, encriptación de la base de datos, **soporte al driver ODBC**, etc.

Podemos observar que **la base de datos H2** es una muy buena opción para agregar una base de datos embebida, en memoria, o de tipo cliente/servidor a nuestro aplicativo. Es por ello que, a continuación, vamos a realizar

la instalación de una base de datos H2 en una aplicación web Java ayudándonos del Framework, ya conocido Spring Boot.

¿Qué es Spring Boot?

Como ya hemos adelantado en el vídeo, sabemos que *Spring* es un *framework* muy conocido. **Spring Boot es un módulo dentro de Spring**, y, últimamente, está en auge por su usabilidad y el hacer fácil ciertas partes del proceso que por defecto son más complejas.

Por ejemplo, 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. Pues bien, *Spring Boot* nace con el objetivo de facilitar al máximo la primera parte (selección de dependencias) y la última (despliegue de la aplicación).

/ 7. Base de datos embebida en aplicación web a través de Spring Boot I: Añadir BBDD

Tal y como hemos comentado en el punto anterior, usaremos Spring Boot, ya que es una herramienta que nos facilitará muchos aspectos a lo largo de nuestro aprendizaje. En este punto, nos será de gran utilidad añadir una base de datos que facilite el almacenamiento de información para nuestro aplicativo web.

En primer lugar, nos dirigiremos a la web https://start.spring.io/, que nos ofrece un entorno sencillo para crear el arquetipo de nuestra aplicación con distintas variables. Veamos:

- En la primera opción **project,** elegiremos **maven**. Ya que nuestro ejemplo será implementado y compilado con dependencias *maven*.
- En la segunda opción **Spring boot**, dejaremos la release version que venga por defecto, ya que será la más estable por el momento.
- En la opción **project metadata**, tendremos las siguientes opciones:
  - **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 la parte final de **Dependencies** será donde incluiremos nuestra base de datos H2 embebida. Esta parte, realmente, es una herramienta muy potente, ya que nos permite incluir la mayoría de los *framework* y utilidades que rodean el lenguaje por medio de dependencias que serán incluidas automáticamente.

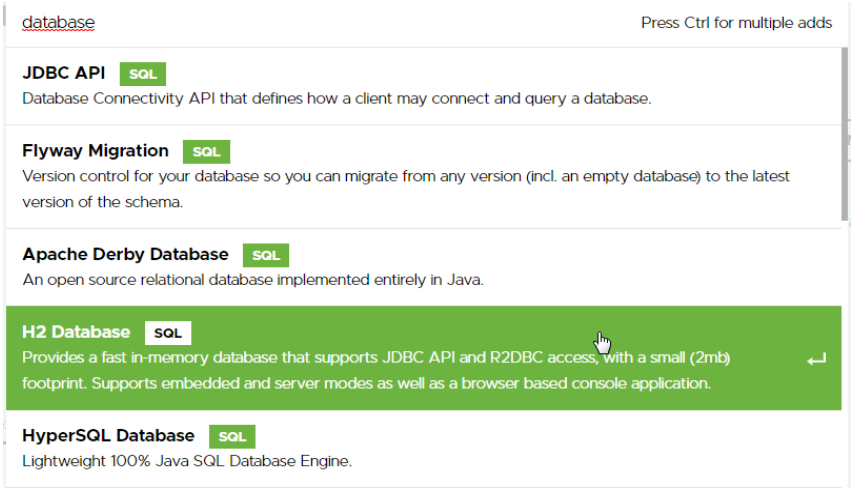

Fig. 8. Diferentes bases de datos embebidas.

En este apartado, incluiremos nuestra base de datos, pero como podemos observar, están presentes las anteriormente mencionadas.

/ 8. Base de datos embebida en aplicación web a través de Spring boot II: IDE

Hemos añadido la base de **datos H2** y el módulo *"Spring Web"* (ya que suponemos que vamos a simular la funcionalidad de una aplicación web cliente/servidor). Una vez añadidos los módulos que necesitemos para nuestro aplicativo, hacemos clic en "GENERATE" y automáticamente se **nos descargará un fichero con extensión .zip o .rar**, y en su interior, tendremos un proyecto web java con todas las características que hemos indicado.

Una vez tengamos nuestro fichero y descomprimido el directorio del proyecto, procedemos a **abrirlo desde el IDE** que usemos frecuentemente, sea Intellij, Eclipse o cualquiera de ellos. Cuando agreguemos el proyecto, podemos ir al "pom.xml" principal y ver como 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>
```

Código 1. Dependencia H2.

Como podemos ver, las librerías de la base de datos H2 han sido añadidas y perfectamente integradas por Spring Boot.

Por defecto, **Spring Boot nos genera un fichero en el proyecto llamado "Application.properties"** donde tenemos algunos aspectos de configuración básica de nuestra base de datos H2:

```
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
```

Código 2. Config. H2.

En la primera línea, 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;
```

Código 3. Config. 2 H2.

De esta forma, nuestra base de datos quedará almacenada en la ruta que se indica.

/ 9. Base de datos embebida en aplicación web a través de Spring Boot III: Inicio BBDD

Nuestra base de datos está montada y preparada para funcionar. Evidentemente, sería ideal montar una buena **capa DAO 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. Nosotros, para hacer un test de la misma, vamos a indicar otros ficheros de configuración.

A continuación, nos dirigimos a la ruta "src/main/resources". Una vez ahí, 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:

```
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'
);
```

Código 4. Inicio base datos.

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. Esto es tan sencillo como añadir en nuestro fichero "application.properties" la siguiente línea:

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

Código 5. Habilitando consola.

De esta forma, cuando nuestra aplicación esté ejecutándose, nos dirigiremos a nuestro navegador e introduciremos la siguiente URL: http://localhost:8080/h2-console.

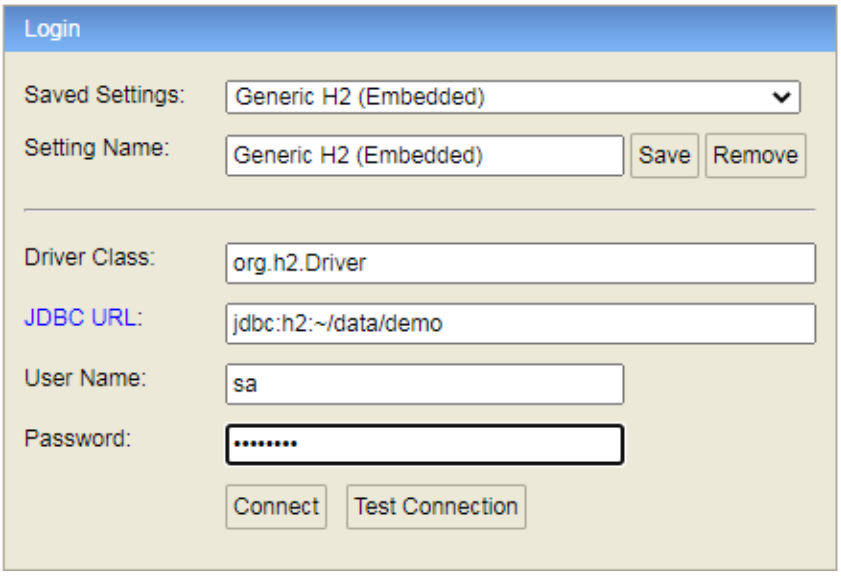

Fig. 9. Acceso a panel de control.

De esta forma, accederemos a nuestro panel de control de la base de datos, donde podremos realizar distintas operaciones. Podremos verlo en el siguiente vídeo:

Vídeo 2. "Panel de control"
https://bit.ly/2B4M7xa

/ 10. Gestores de base de datos independientes

Una vez vistos **los sistemas de bases de datos embebidos**, repasaremos **los sistemas independientes** y qué diferencias notables existen entre ambas opciones.

Quizás, una de las diferencias más importantes es que una base de datos independiente **no puede ejecutarse bajo la misma máquina virtual** de Java que usa nuestra aplicación en ejecución. Una base de datos independiente 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.

Una base de datos independiente **consumirá siempre más recursos** que una embebida, es por ello que al estar en una máquina aislada para dicha base de datos, aprovechará mejor los recursos que tiene disponibles. A nivel de funcionalidad y operativo, las bases de datos embebidas están restringidas, sin embargo, las bases de datos independientes se preparan en dicha máquina separada y aislada para volcar todo el potencial que ofrece con todos sus procedimientos y operativos.

Las bases de datos relacionales independientes más conocidas a nivel comercial y de mayor potencia y extensión:

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

Fig. 10. Logos de las BBDD independientes más conocidas.

/ 11. Caso práctico 2: "Cambio de credenciales"

Planteamiento: Alguien del equipo de desarrollo, en una tarea de mantenimiento del aplicativo, ha cambiado la tabla de usuarios de acceso al panel de la base de datos de una web de reservas.

Ahora, para el usuario "user" la contraseña es "1234Pass".

Nos solicitan un cambio urgente en la base de datos embebida H2 que tiene la aplicación, ya que actualmente está fallando en los entornos de producción.

Solucionar el problema lo antes posible es de vital importancia para el cliente.

Nudo: Una vez modificada la base de datos (la base de información donde se alojan los datos de acceso), habría que investigar en el entorno de la aplicación qué ficheros de configuración existen para el tema de conexiones y qué claves habría que cambiar.

Desenlace: Realmente, como estamos usando el sistema Spring Boot, el cual lleva incluido Maven para la gestión de

dependencias y compilación de las mismas, y tenemos embebida una base de datos de tipo H2, la primera idea debe de ser irnos a la paquetería de recursos de nuestra aplicación:

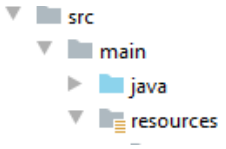

Fig. 11. Resources.

Una vez ahí, intentamos localizar un fichero con extensión .properties, en el cual estarían las siguientes claves:

```
spring.datasource.username=user
spring.datasource.password=1234Pass
```

Código 6. Credenciales.

Simplemente cambiando aquí las credenciales con los nuevos valores establecidos en la tabla de base de datos, volveremos a tener conexión con nuestra base de datos.

/ 12. Resumen y resolución del caso práctico de la unidad

En esta unidad, hemos visto las **diferencias entre una base de datos embebida y una independiente.** Profundizando en ellas, hemos podido comprobar la variedad de bases de datos embebidas que existen para nuestro aplicativo, ya sean sobre el disco o en memoria del sistema.

Se han presentado los **diferentes registros de base de datos embebidas** y se ha mostrado **la instalación y configuración de la base de datos H2** mediante el ya bien conocido *framework* **Spring Boot** para un sistema de almacenado en una aplicación web.

Por último, se han repasado las **diferencias notables entre bases de datos embebidas e independientes**, y se han mostrado las **bases de datos independientes más usadas.**

Resolución del caso práctico inicial

En relación a la tarea que se le encarga a Juan, decir, en primer lugar, que depende mucho del tipo de aplicativo al que esté vinculada esta base de datos. Si tiene mucha afluencia de datos, de transacciones, de consultas, etc., en este caso, elegiríamos una base de datos **independiente**; si, por el contrario, es una base de datos con pocas tablas y de no tanta afluencia de acceso a la capa de datos, podríamos incluir una embebida de almacenamiento en disco, tal y como hemos visto en la unidad.

Pongamos el caso que el ejemplo del aplicativo de Juan es sencillo y, por lo tanto, con una base de datos **embebida** nos serviría. En este caso, tendríamos dos opciones:

1. Descargar el .jar de la página oficial de Apache Derby: https://db.apache.org/derby/derby_downloads.html#For+Java+8+and+Higher. De esta forma, podremos añadir este fichero a nuestro aplicativo, el cual contiene el motor de la base de datos y el conector.

2. Una forma más desatendida sería, directamente, montar nuestro proyecto con *Spring Boot*, dirigirnos a la página spring.io, como vimos en el punto anterior, y en el siguiente punto de la imagen, elegir la dependencia de Derby DB.

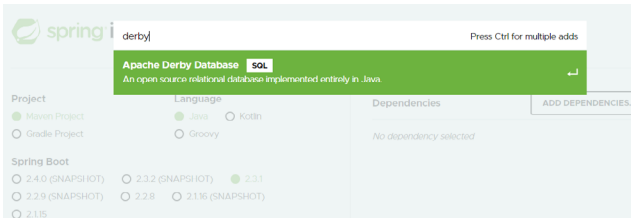

Fig. 12. Derby DB.

/ 13. Webgrafía

https://start.spring.io/