


Open · Opened 2 days ago by  **Rubén Montero**

Mi primera conexión a una base de datos

Resumen

- Hablaremos de la evolución de las bases de datos
- Comprenderemos que hay distintos tipos de *sistemas gestores de bases de datos* (SGBD)
- Escribiremos una clase nueva en Java que se conectará a una base de datos mediante un *conector*
- (*Y como siempre, subiremos los cambios al repositorio*)

Descripción

Las [bases de datos](#) son una parte esencial de todo sistema informático. ¿Haces una compra *online* y esperas que el pedido esté en el historial? ¿Inicias sesión en Twitter y ves tus *tweets* marcados como favoritos? ¿Realizas un trámite en la [sede de la Xunta](#) y finalmente llega el resultado del procedimiento? ¿Juegas a tu título favorito de [PlayStation](#) (¿o [Xbox](#)? ¿[Switch](#)?) y la partida sigue donde la dejaste? **Claro. Nosotros, desarrolladores**, escribimos **programas** que procesan datos de forma **automatizada**.

Pero, ¿*dónde* se **almacenan** esos **datos**?

Para no tenerle miedo a las *bases de datos*, lo mejor es entender que surgen como evolución natural de nuestras necesidades como desarrolladores.

1. Los lenguajes de programación y entornos de desarrollo eran muy distintos décadas atrás
2. Para las necesidades de una aplicación, probablemente se guardase información en ficheros (*¡como nosotros en el proyecto anterior!*) a gusto del programador
3. Así que cada programa vivía en su pequeño reino e intercambiar o reutilizar datos se volvía complicado
4. Además surgían problemas de inconsistencias, duplicidades...

La evolución natural: SQL

[Structured Query Language \(SQL\)](#) es un lenguaje que viene de la mano del [modelo relacional](#) de [Frank Codd](#).

La idea es muy sencilla:

- Guardar los datos en forma de **tablas** (*modelo relacional*)
- Tener un **lenguaje estandarizado** para procesar los datos (*SQL*)

¿Pero cómo exactamente se guardan los datos?

¡Ah! Depende.

Eso, el bueno de [Frank](#) no nos lo dijo. Tan sólo podemos afirmar que SQL pasó a ser el estándar [ANSI](#) en 1986, e [ISO](#) en 1987.

¿**Cómo** implementar el estándar? ¿**Cómo** guardar los datos?

Depende.

Es decir, los **programas** que soportan **consultas SQL** y almacenan y entregan los datos, cada uno a su manera, son muchos y muy variados. Tienen un nombre:

Sistemas gestores de bases de datos (SGBD)

[MySQL](#), [MariaDB](#), [SQLite](#), [PostgreSQL](#) o [SQLServer](#) son algunos ejemplos de *sistemas gestores de bases de datos*.

Nosotros trabajaremos principalmente con **SQLite** por su simplicidad.

Comencemos

Como primer paso, abre un terminal (cmd.exe) en tu ordenador y cambia el directorio activo hasta la ubicación de tu repositorio usando `cd`. Haz:

```
git pull
```

...para traer los cambios de remoto a local, que contienen correcciones del proyecto anterior y **el esqueleto del nuevo proyecto**.

Se encontrará en **una carpeta llamada `java-rest`**.

Abre el proyecto desde [IntelliJ IDEA](#) y verás un entorno con el que ya eres familiar.

Creando la primera clase

Dentro de `src/`, crea una nueva clase llamada `MoviesDataProvider`.

La usaremos para conectarnos a una base de datos local SQLite.

Un momento, un momento...

¿Conectarnos?

Sí. La idea central de este proyecto es **acceder programáticamente** (mediante código) a bases de datos.

Es común acceder a una base de datos usando un programa cliente como [MySQL Workbench](#). Sin embargo, también podemos emplear un **conector** para que esta conexión **no** sea *manual*, sino que se produzca **desde código Java**.

Interesante... ¿Qué es un conector?

Es una clase Java que se encarga de realizar la conexión.

Estas clases **no** las escribimos nosotros, sino los desarrolladores del SGBD correspondiente.

En nuestro proyecto `java-rest/`, dentro de `lib/`, **ya** están añadidos los conectores necesarios para que se puedan establecer conexiones a:

- Bases de datos SQLite ([sqlite-jdbc-3.36.0.3.jar](#))

Estos conectores son fácilmente accesibles buscando los `.jar` en Internet, o añadiendo las dependencias al proyecto si usamos un sistema automatizado como [Maven](#) o [Gradle](#).

¿Cómo se realiza la conexión?

Hay que *invocar* el método [estático](#) `.getConnection` de la clase `DriverManager` del paquete `java.sql`. Viene a ser un [patrón factoría](#).

Si todo sale bien, se devolverá una *instancia* del tipo `Connection`:

```
Connection conn = DriverManager.getConnection(/* Aquí se describe la base de datos */);
```

Nótese que tanto `DriverManager` como `Connection` son clases estándar de Java. Podemos usarlas *aunque no hayamos añadido* los `.jar` de los conectores arriba mencionados.

Entonces, ¿los conectores son necesarios realmente?

Sí.

En `.getConnection` **pediremos** una conexión a un SGBD en particular, y si **no** está presente el conector necesario, saltará una **excepción**. También, si falla la conexión.

Así que hay que controlar dicha **excepción** `SQLException`:

```
try {
    Connection conn = DriverManager.getConnection(/* Aquí se describe la base de datos */);
} catch (SQLException e) {
    throw new RuntimeException(e);
}
```

String para describir una conexión

A `.getConnection` debemos pasarle un `String` (" ") de la siguiente forma:

```
"jdbc:<tipo de SGBD>:<parámetros>"
```

Conectarse a SQLite

Cada SGBD necesita diferentes parámetros.

[SQLite](#) es un SGBD **ultra-ligero**. Básicamente, **toda** la base de datos se almacena en un **fichero** en el disco duro. Por lo tanto, para conectarnos a una base de datos **SQLite** sólo necesitamos conocer la **ruta** al fichero de base de datos.

¡Qué fácil!

Sí.

¿Y cómo?

Usando una ruta relativa:

```
"jdbc:sqlite:unaCarpeta/otraCarpeta/miBaseDeDatos.db"
```

o absoluta, con esta sintaxis:

```
"jdbc:sqlite:C:/carpetaEnRaiz/otraCarpeta/miBaseDeDatos.db"
```

También se permiten bases de datos en memoria RAM usando SQLite (`"jdbc:sqlite::memory:"`), aunque no lo usaremos.

Volvamos a **MoviesDataProvider**

En nuestro ejemplo inicial, nos conectaremos a una base de datos SQLite que reside en:

- `db/sqlite3/`

...dentro del propio proyecto. Se llama `movies.db`.

El código inicial en `MoviesDataProvider` sería:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MoviesDataProvider {

    // Método constructor
    public MoviesDataProvider() {
        String connectionStr = "jdbc:sqlite:db/sqlite3/movies.db";
        try {
            Connection conn = DriverManager.getConnection(connectionStr);
            // ...
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Ahora, añade un `System.out.println` después de que se establezca la conexión. Imprimirá `"Connection established"`.

Si creas una nueva instancia desde `Main.java`:

```
public class Main {
    public static void main(String[] args) {
        MoviesDataProvider provider = new MoviesDataProvider();
    }
}
```

...y añades una configuración (barra superior > Add Configuration) para `Main`; al darle al botón verde 'Play' debería ejecutarse la aplicación y mostrar el `print` en pantalla.

¡Enhorabuena! Has abierto tu primera conexión automática a un sistema gestor de base de datos mediante un conector Java.

Por último

Verifica que el test funciona correctamente.

Haz `commit` y `push` para subir los cambios al repositorio.

No está de más visitar la página de GitLab y verificar que el `commit` se ha subido.



Rubén Montero @ruben.montero changed milestone to [%Sprint 2](#) 2 days ago

