Tema 3. Tecnologías del servidor. Bases de datos: Spring Data

Programación web

Boni García

Curso 2017/2018

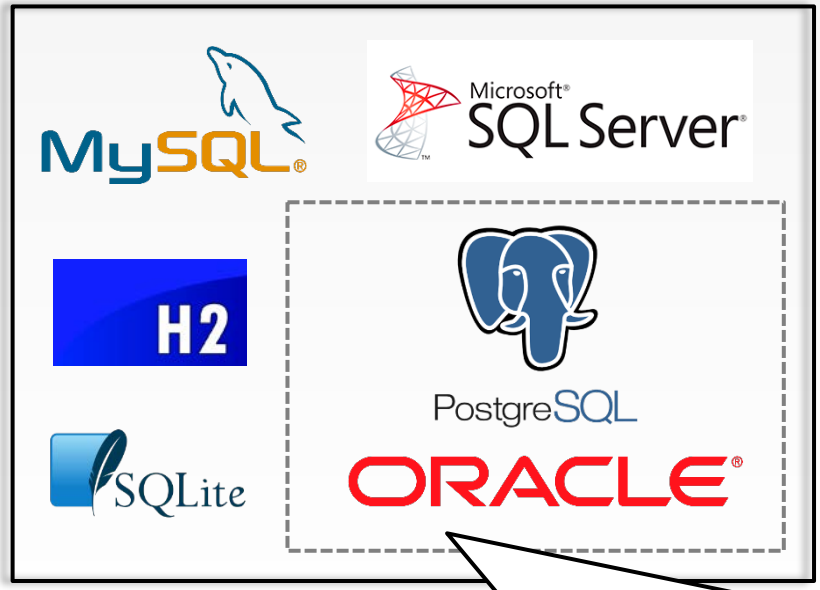
Índice

Índice

Introducción

- Un sistema gestor de bases de datos (**DBMS**) es el software que permite almacenar y consultar bases de datos (conjunto de datos)

Bases de datos relacionales (RDBMS)

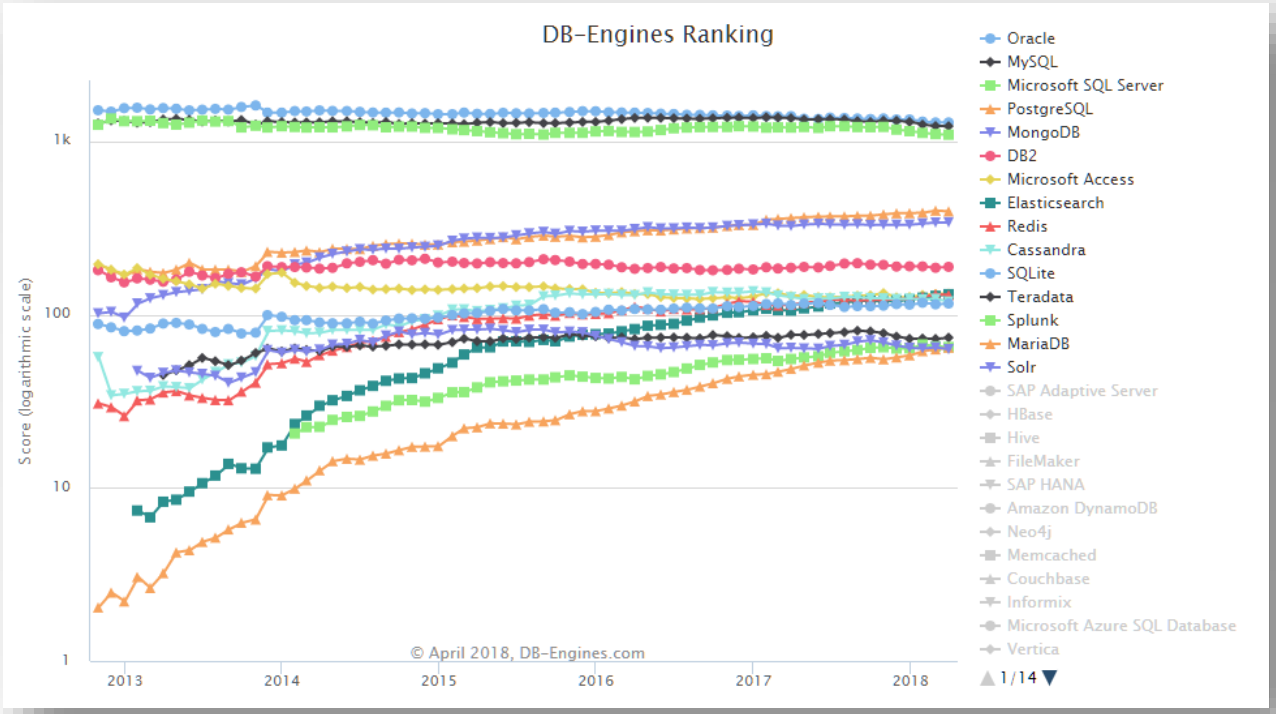


Bases de datos objeto-relacionales (ORDBMS)

Bases de datos “no sólo SQL” (NoSQL)

Introducción

- Ranking de uso de bases de datos:



http://db-engines.com/en/ranking_trend

Bases de datos relacionales

Introducción

- Una **base de datos relacional** almacena la información en tablas relacionadas (mediante clave primarias y foráneas)
- **SQL** (*Standard Query Language*) es un lenguaje diseñado para gestionar una base de datos relacional. Tipos comandos SQL:
  - Lenguaje de Manipulación de Datos (DML): Obtiene, Inserta, Borra y actualiza datos (`SELECT`, `INSERT`, `DELETE`, `UPDATE`)
  - Lenguaje de Definición de Datos (DDL): Crea, borra y cambia tablas, usuarios, vistas, índices… (`CREATE TABLE`, `DROP TABLE`, `ALTER TABLE`)
- **JDBC** (*Java DataBase Connectivity*) es la API estándar de acceso a base de datos desde Java (en Java SE 8 se incluye JDBC 4.2)

Bases de datos relacionales

JDBC

- Ejemplo: proyecto Maven para acceder a base de datos MySQL

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>io.github.web</groupId>
   <artifactId>jdbc</artifactId>
   <version>1.0.0</version>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <java.version>1.8</java.version>
   </properties>

   <dependencies>
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <version>5.1.46</version>
         <scope>runtime</scope>
      </dependency>
   </dependencies>
</project>
```

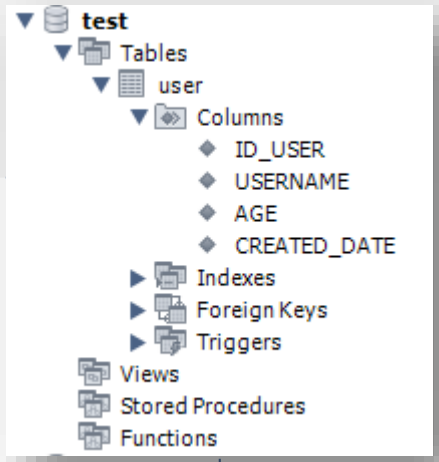
Driver JDBC para MySQL

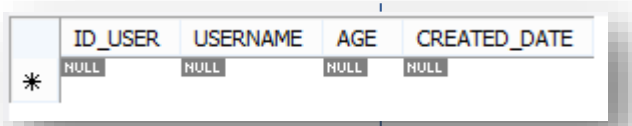
Bases de datos relacionales

JDBC

- Ejemplo: creación de tabla

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

public class MySqlJdbcCreateTable {
   public static void main(String[] args) throws Exception {
      // MySQL JDBC Driver
      Class.forName("com.mysql.jdbc.Driver");
      // Connection to MySql
      Connection connection = DriverManager.getConnection(
         "jdbc:mysql://localhost:3306/test", "root", "");
      // Create table
      String createSql = "CREATE TABLE USER (ID_USER INT NOT NULL AUTO_INCREMENT, "
         + "USERNAME VARCHAR(45) NULL, AGE INT NULL, "
         + "CREATED_DATE DATE NOT NULL, PRIMARY KEY (ID_USER))";
      Statement statement = connection.createStatement();
      statement.execute(createSql);
      statement.close();
      // Close connection
      connection.close();
   }
}
```

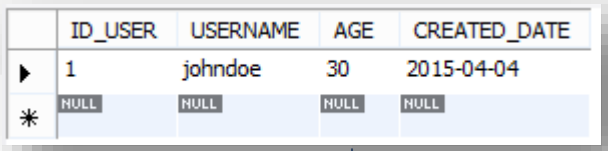




Bases de datos relacionales

JDBC

- Ejemplo: insertar datos en la tabla



	ID_USER	USERNAME	AGE	CREATED_DATE
▶	1	johndoe	30	2015-04-04
*	NULL	NULL	NULL	NULL

```
// MySQL JDBC Driver
Class.forName("com.mysql.jdbc.Driver");

// Connection to MySql
Connection connection = DriverManager.getConnection(
   "jdbc:mysql://localhost:3306/test", "root", "");

// Insert row
java.util.Date now = new java.util.Date();
java.sql.Date sqlDate = new java.sql.Date(now.getTime());
String insertSql = "INSERT INTO USER (USERNAME, AGE, CREATED_DATE) "
   + "VALUES ('johndoe', 30, '" + sqlDate + "')";
Statement statement = connection.createStatement();
statement.execute(insertSql);
statement.close();

// Close connection
connection.close();
```

Bases de datos relacionales

JDBC

- Ejemplo: leer y modificar datos (presuponemos la conexión abierta):

```
// Read row
String selectSql = "SELECT ID_USER FROM USER WHERE USERNAME='johndoe'";
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery(selectSql);
rs.last();
int id = rs.getInt("ID_USER");
rs.close();
statement.close();

// Update row
String updateSql = "UPDATE USER SET AGE=35 WHERE ID_USER=" + id;
statement = connection.createStatement();
statement.execute(updateSql);
statement.close();
```

	ID_USER	USERNAME	AGE	CREATED_DATE
▶	1	johndoe	35	2015-04-04
*	NULL	NULL	NULL	NULL

Bases de datos relacionales

JDBC

- Ejemplo: borrar datos (presuponemos la conexión abierta):

```
// Delete row
String deleteSql = "DELETE FROM USER WHERE ID_USER=" + id;
statement = connection.createStatement();
statement.execute(deleteSql);
statement.close();
```

Bases de datos relacionales

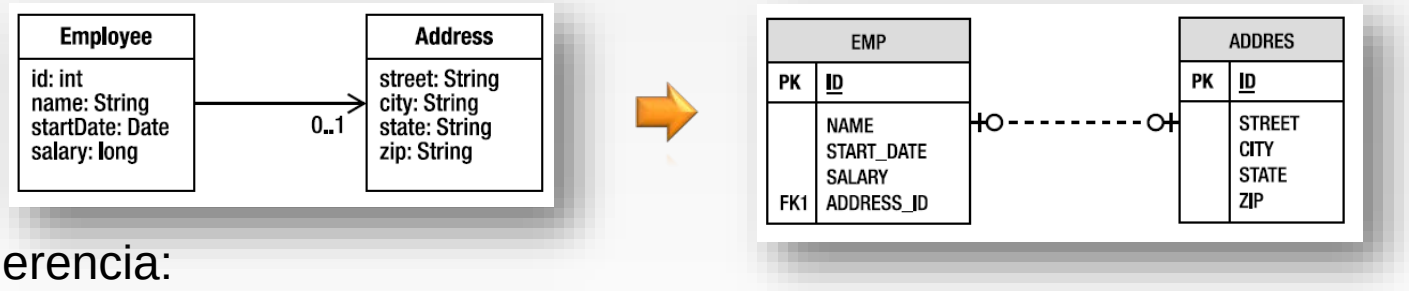
JPA

- La técnica para convertir datos del sistema de tipos de un lenguaje **orientado a objetos** y el **modelo relacional** de las bases de datos se conoce como mapeo objeto relacional (**ORM**, *Object Relational Mapping*)
  - Generación de tablas partiendo de clases
  - Generación de clases partiendo de tablas
- **JPA** (*Java Persistence API*) es la especificación de ORM para Java
- JPA internamente usa JDBC
- Hay diferentes implementaciones JPA:
  - Hibernate: http://hibernate.org/
  - Toplink: http://www.oracle.com/technetwork/middleware/toplink/overview/index.html
  - Spring Data JPA: https://projects.spring.io/spring-data-jpa/

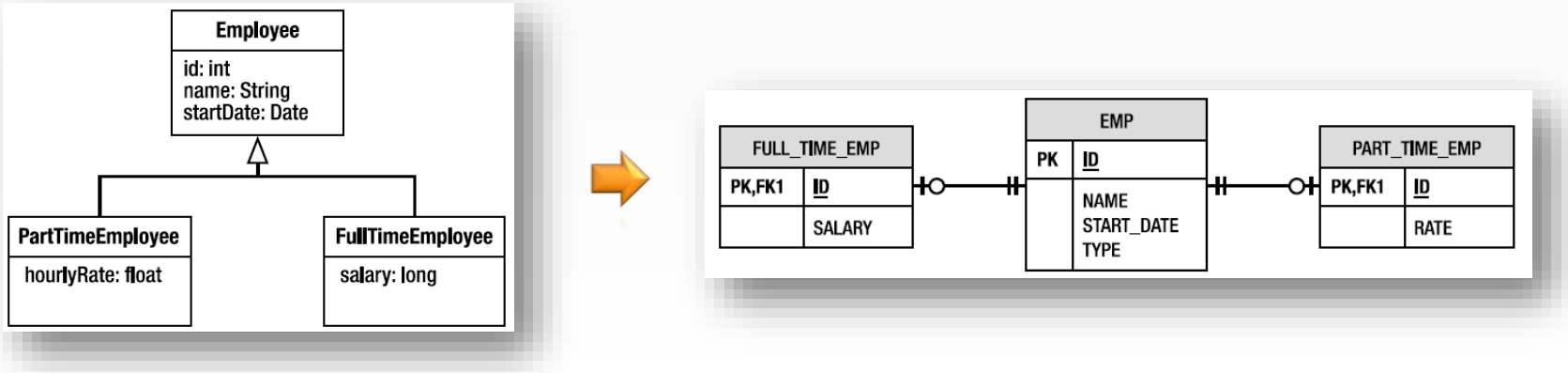
Bases de datos relacionales

JPA

- Correspondencias básicas objetos/tablas
  - Relación:



  - Herencia:



Bases de datos relacionales

Spring Data

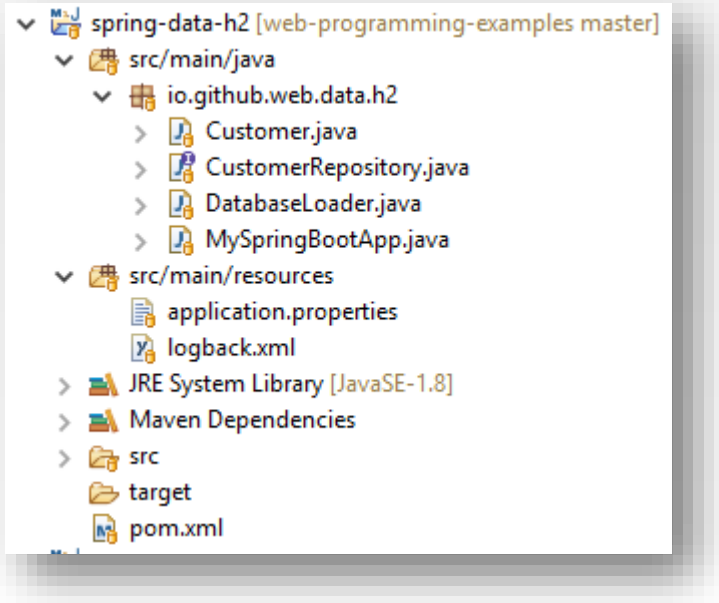
- El proyecto **Spring Data** ofrece mecanismos para simplificar el acceso a bases de datos
- Nosotros vamos a estudiar **Spring Data JPA** y **Spring Data MongoDB** en aplicaciones Spring Boot
- Ambos sub-proyectos ofrecen las siguientes capacidades:
  - **Conversión automática** entre objetos Java persistentes (llamados **entidades**) y el esquema de la base de datos al inicio de la aplicación
  - **Creación de consultas** en base a métodos declarados en interfaces Java

http://projects.spring.io/spring-data/

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

- Pasos para implementar una aplicación Spring Data JPA / Boot
  1. Configurar `pom.xml`
  2. Crear objetos de **entidad**
  3. Crear **consultas** a la base de datos
  4. Hacer uso de base de datos
  5. Ejecutar la aplicación



Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

1. Configurar `pom.xml`

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
</dependencies>
```

Spring Boot

Aplicación web

Spring Data JPA

Base de datos H2

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

2. Crear objetos de **entidad** (que serán mapeados en la base de datos)

```
@Entity
public class Customer {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;

   private String firstName;
   private String lastName;

   // Default contructor (needed by SpringData)
   protected Customer() {
   }

  public Customer(String firstName, String lastName) {
     this.firstName = firstName;
     this.lastName = lastName;
   }

   // Getter, Setters and toString
}
```

Al anotar una clase como `@Entity` estamos indicando a JPA que se trata de un objeto que tendrá su equivalente en la base de datos

El atributo anotado con `@Id` será la clave primaria (en este caso también será auto incremental)

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

3. Crear **consultas** a la base de datos
   - Vamos a crear consultas creando interfaces que extienden de la clase `CrudRepository<[Entity],[PrimaryKey]>`
   - Las consultas se realizan mediante métodos con nombre `findBy*`
   - Cada método se traducirá automáticamente en consultas a la base de datos

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

   List<Customer> findByLastName(String lastName);

   List<Customer> findByFirstName(String firstName);

}
```

Al extender de `CrudRepository` automáticamente dispondremos de métodos:

- `save(Customer)`
- `delete(Customer)`
- `delete(Long)`
- `findOne(Long)`
- `findAll()`

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

3. Crear **consultas** a la base de datos
   - Algunas palabras clave usados en el nombre de los métodos:

Keyword	Ejemplo
And	findByLastnameAndFirstname
Or	findByLastnameOrFirstname
Equals	findByFirstname
LessThan	findByAgeLessThan
LessThanEqual	findByAgeLessThanEqual

Keyword	Ejemplo
IsNull	findByAgeIsNull
StartingWith	findByFirstnameStartingWith
EndingWith	findByFirstnameEndingWith
Containing	findByFirstnameContaining
IgnoreCase	findByFirstnameIgnoreCase

   - Referencia: http://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

3. Crear **consultas** a la base de datos

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);

    List<Customer> findByFirstName(String firstName);

    @Query(value = "SELECT * FROM CUSTOMER", nativeQuery = true)
    List<Customer> selectAll();

}
```

También podemos usar SQL directamente usando la anotación `@Query`

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

4. Hacer uso de base de datos

Hacemos uso de la **inyección de dependencias** Spring para usar el repositorio donde sea necesario

En este ejemplo creamos un componente Spring y anotamos un método con `@PostConstruct` para que se ejecute este código justo después de la creación del componente

```
@Component
public class DatabaseLoader {
    private final Logger log = LoggerFactory.getLogger(this.getClass());
    private CustomerRepository repository;
    public DatabaseLoader(CustomerRepository repository) {
        this.repository = repository;
    }
    @PostConstruct
    private void initDatabase() {
        // Create
        repository.save(new Customer("John", "Doe"));
        repository.save(new Customer("Michael", "Smith"));
        // Update
        Customer firstCustomer = repository.findAll().iterator().next();
        firstCustomer.setFirstName("Peter");
        log.info("Updating {}", firstCustomer);
        repository.save(firstCustomer);
        // Read
        Iterable<Customer> all = repository.findAll();
        for (Customer customer : all) {
            log.info("Reading {}", customer);
        }
        // Delete
        long firstId = repository.findAll().iterator().next().getId();
        repository.deleteById(firstId);
        log.info("Number of costumer(s) after deleting: {}",
                repository.count());
    }
}
```

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

5. Ejecutar la aplicación

```
@SpringBootApplication
public class MySpringBootApp {

  public static void main(String[] args) {
    SpringApplication.run(MySpringBootApp.class, args);
  }
}
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

...
INFO 11236 --- [main] io.github.web.data.h2.DatabaseLoader : Updating Customer [id=1, firstName=Peter, lastName=Doe]
INFO 11236 --- [main] io.github.web.data.h2.DatabaseLoader : Reading Customer [id=1, firstName=Peter, lastName=Doe]
INFO 11236 --- [main] io.github.web.data.h2.DatabaseLoader : Reading Customer [id=2, firstName=Michael, lastName=Smith]
INFO 11236 --- [main] io.github.web.data.h2.DatabaseLoader : Number of costumer(s) after deleting: 1
...
```

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

- El comportamiento por defecto de una base de datos H2 es funcionar como base de datos en memoria
- Nos puede interesar almacenar los datos de forma persistente en el sistema de ficheros
- Para ello hay que configurar el fichero `application.properties`

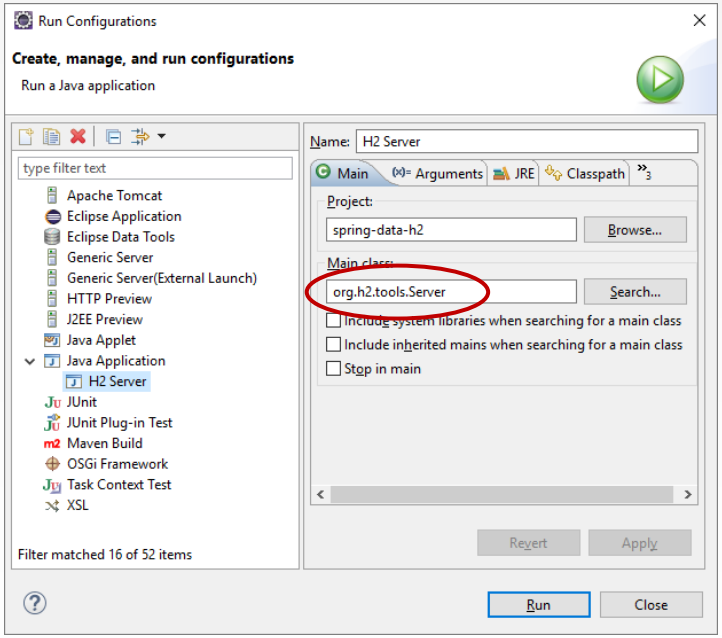
```
# Uncomment this line to store H2 in local file system
spring.datasource.url=jdbc:h2:~/test;AUTO_SERVER=TRUE
```

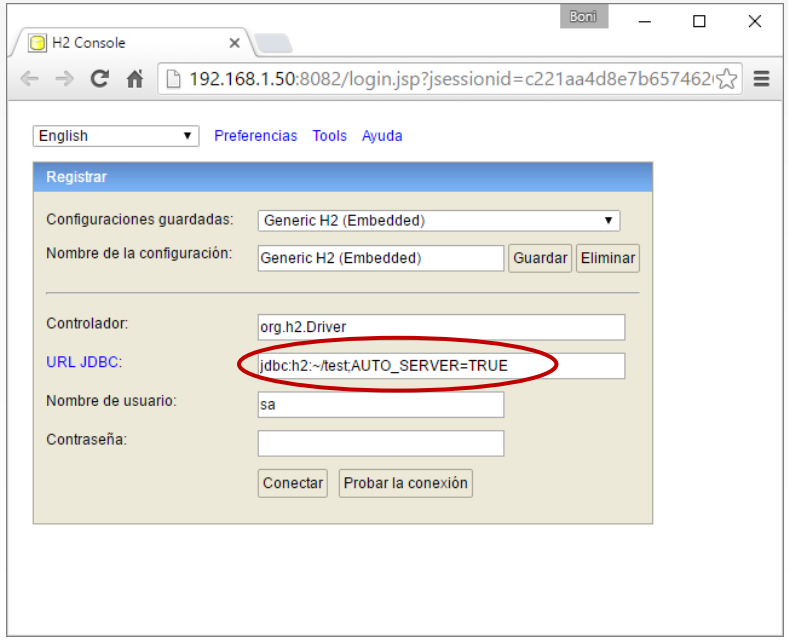
- Más información: http://www.h2database.com/html/features.html

Bases de datos relacionales

Spring Data JPA con base de datos relacional H2

- También nos puede interesar hacer consultas en una base de datos H2
- Para ello podemos usar el frontal web que trae por defecto H2

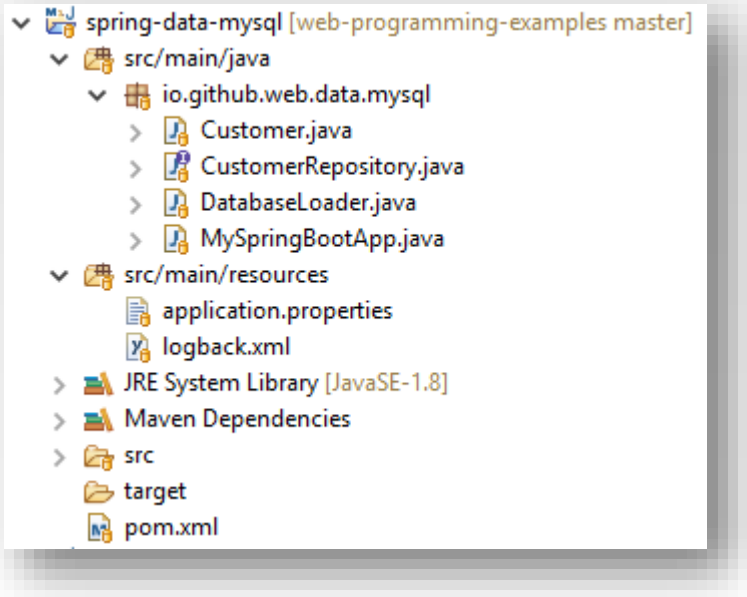




Bases de datos relacionales

Spring Data JPA con base de datos relacional MySQL

- En las bases de datos persistentes (MySQL, Oracle…) hay que gestionar adecuadamente la **creación del esquema**
- Vamos a partir del ejemplo anterior (con H2) y lo vamos a modificar para usar una base de datos MySQL



Bases de datos relacionales

Spring Data JPA con base de datos relacional MySQL

- En primer lugar hay que modificar el `pom.xml`

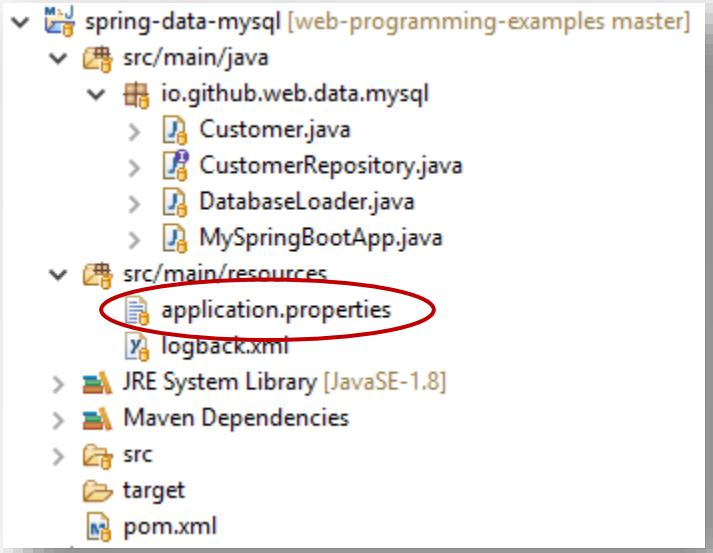
```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.0.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
    </dependencies>
```

Simplemente hay que cambiar la base de datos, esta vez usamos MySQL

Bases de datos relacionales

Spring Data JPA con base de datos relacional MySQL

- En segundo lugar hay que modificar el `application.properties`




```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create-drop
```

Con esta configuración la base de datos MySQL deberá estar arrancada en la máquina local y deberemos tener acceso con el usuario `root` (sin contraseña en este ejemplo)

Bases de datos relacionales

Spring Data JPA con base de datos relacional MySQL

- Para la gestión del esquema jugaremos con el valor de la propiedad `spring.jpa.hibernate.ddl-auto`
  - `spring.jpa.hibernate.ddl-auto=none`: No hace nada con el esquema
  - `spring.jpa.hibernate.ddl-auto=validate`: Verifica que el esquema de la base de datos es compatible con las entidades de la aplicación y si no lo es genera un error
  - `spring.jpa.hibernate.ddl-auto=update`: Incluye en el esquema actual los elementos necesarios para hacer el esquema compatible con las entidades (no borra ningún elemento)
  - `spring.jpa.hibernate.ddl-auto=create-drop`: Crea el esquema al iniciar la aplicación y le borra al finalizar (igual que una BBDD en memoria)

Bases de datos relacionales

Spring Data JPA con base de datos relacional MySQL

- Cuándo usar los diferentes tipos de gestión de esquema:
  - `create-drop`: En desarrollo
  - `validate`: En desarrollo, usando un esquema existente
  - `update`: Cuando queramos crear el esquema en la base de datos a partir de las entidades (clases Java) que hemos definido
  - `none`: En producción

Bases de datos NoSQL

MongoDB en Java

- MondoDB driver:

```
<dependency>
   <groupId>org.mongodb</groupId>
   <artifactId>mongo-java-driver</artifactId>
   <version>3.6.3</version>
</dependency>
```

- Creación cliente:

```
MongoClient mongo = new MongoClient("localhost", 27017);
MongoDatabase db = mongo.getDatabase("test");
MongoCollection<Document> collection = db.getCollection("user");
// Accessing MongoDB
mongo.close();
```

- Creación documento:

```
// Create
Document document = new Document();
document.put("name", "John Doe");
document.put("age", 30);
document.put("createdDate", new Date());
collection.insertOne(document);
```

Bases de datos NoSQL

MongoDB en Java

- Lectura documento:

```
// Read
BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("name", "John Doe");
FindIterable<Document> cursor = collection.find(searchQuery);
System.out.println(cursor.first());
```

- Actualizar documento:

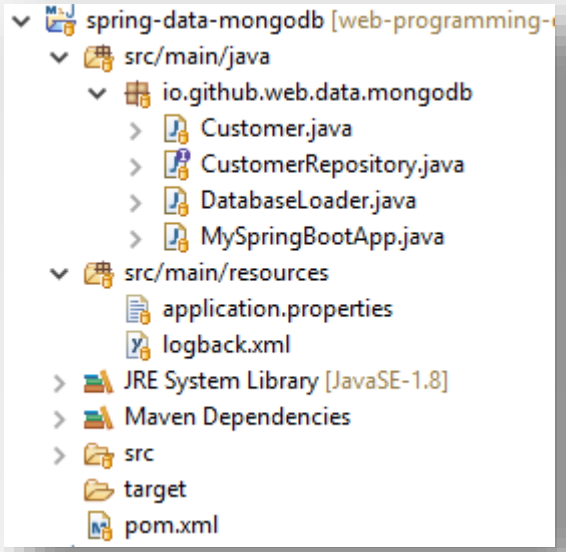
```
// Update
Document documentUpdate = new Document();
documentUpdate.append("$set", new Document("age", 35));
collection.updateOne(searchQuery, documentUpdate);
System.out.println(collection.find(searchQuery).first());
```

- Eliminar documento:

```
// Delete
MongoCursor<Document> iterator = collection.find().iterator();
while (iterator.hasNext()) {
   Document doc = iterator.next();
   collection.deleteOne(doc);
}
```

Bases de datos NoSQL

Spring Data MongoDB



pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
</dependencies>
```

Usamos Spring Data MongoBD en lugar de Spring Data JPA

Bases de datos NoSQL

Spring Data MongoDB

`application.properties`

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
```

El acceso a la base de datos funciona exactamente igual que lo visto anteriormente con H2 y MySQL

La única diferencia es que el identificador incremental en MongoDB es de tipo `String`, no entero

`Customer.java`

```java
import org.springframework.data.annotation.Id;

public class Customer {

   @Id
   private String id;

   private String firstName;
   private String lastName;

   // Default constructor (needed by Spring Data)
   protected Customer() {
   }

   public Customer(String firstName, String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   // Getter, Setters and toString
}
```