INGENIERÍA WEB Y COMPUTACIÓN EN LA NUBE

Bloque3: Parte servidora (backend)

TEMA 3.3: BASES DE DATOS CON SPRING DATA

Boni García
boni.garcia@urjc.es

Universidad Rey Juan Carlos

Índice de contenidos

Índice de contenidos

1. Bases de datos relacionales

SQL (*Standard Query Language*)

- SQL es un lenguaje que sirve para gestionar una base de **datos relacional**
- Los comandos SQL se dividen en categorías:
  - 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`
- Tutorial SQL: http://www.w3schools.com/sql/default.asp

1. Bases de datos relacionales

JDBC

- JDBC (*Java DataBase Connectivity*) es la API estándar de acceso a base de datos desde Java
- Está incluida en Java SE (en Java SE 7 se incluye JDBC 4.1)
- Para conectarse a una base de datos concreta, es necesario su driver JDBC
- El driver es un librería Java que se añade a la aplicación como cualquier otra librería (si usamos Maven, como dependencia)
- La mayoría de las bases de datos incorporan un driver JDBC
- Más información: http://docs.oracle.com/javase/tutorial/jdbc/

1. 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>5.1.35</version>
      </dependency>
   </dependencies>
</project>
```

Driver JDBC para MySQL

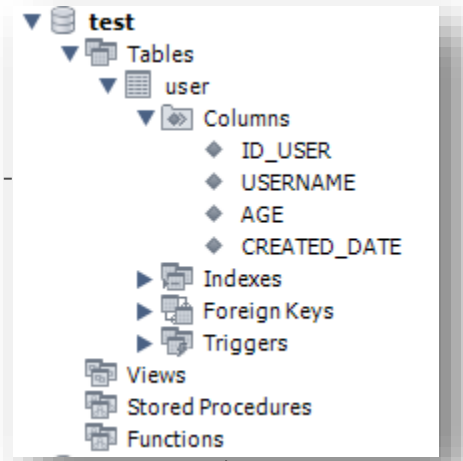
1. Bases de datos relacionales

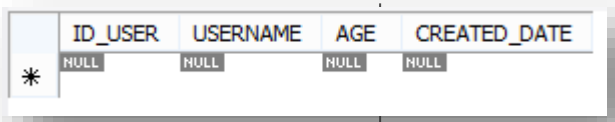
JDBC

- 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();
   }
}
```

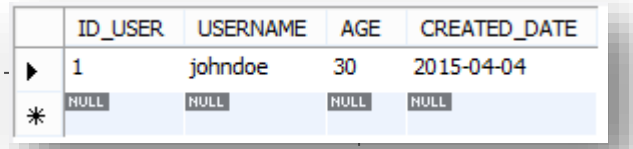




1. Bases de datos relacionales

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();
```

1. 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

1. 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();
```

1. Bases de datos relacionales

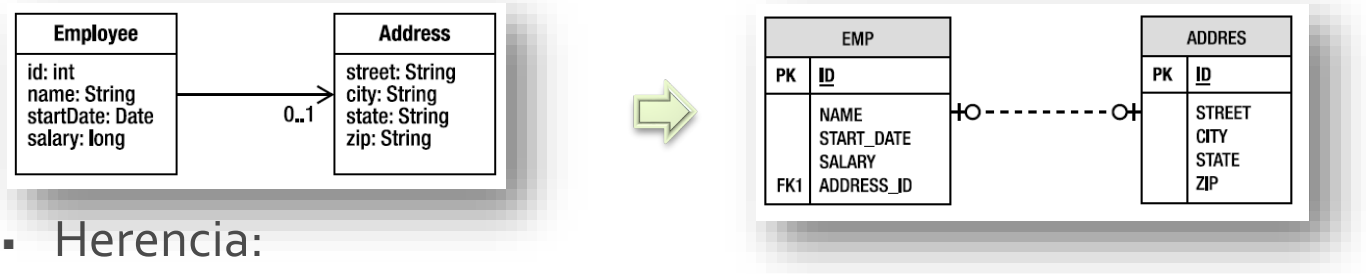
JPA

- 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
- Implementaciones JPA:
  - Hibernate: http://hibernate.org/
  - Toplink: http://www.oracle.com/technetwork/middleware/toplink/overview/index.html
  - ...

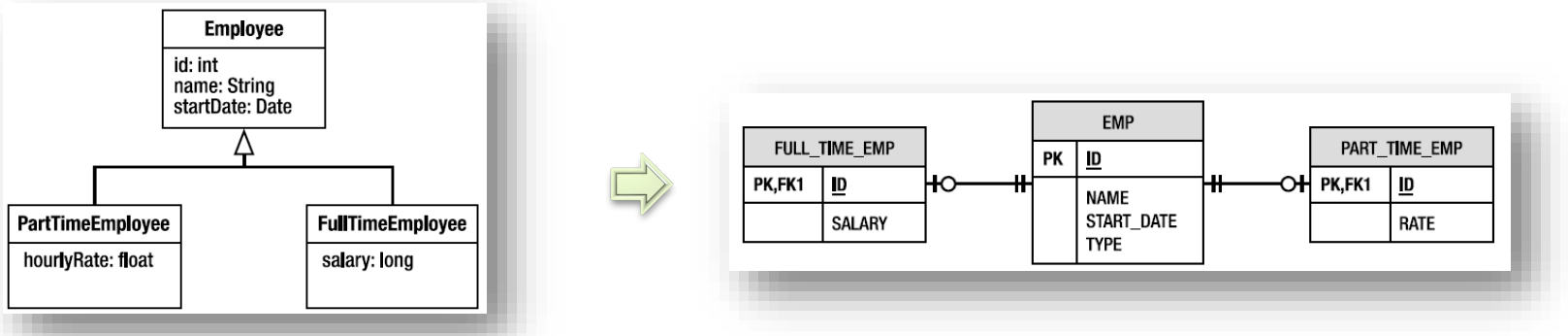
1. Bases de datos relacionales

JPA

- Correspondencias básicas objetos/tablas
  - Relación:



  - Herencia:



https://en.wikibooks.org/wiki/Java_Persistence/Relationships

Índice de contenidos

2. Bases de datos relacionales con Spring

Spring Data JPA

- El proyecto **Spring Data** ofrece mecanismos para simplificar el acceso a diferentes bases de datos:
  - Spring Data JPA
  - Spring Data MongoDB
  - Spring Data JDBC extensions ...
- **Spring Boot** nos permite usar Spring Data de manera más sencilla
- Más información:
  - Spring Data: http://projects.spring.io/spring-data/
  - Spring Boot: http://docs.spring.io/spring-boot/docs/current/reference/html/

2. Bases de datos relacionales con Spring

Spring Data JPA

- Las funcionalidades principales de Spring Data JPA son:
  - **Conversión automática** entre objetos Java y el esquema de la base de datos
  - **Creación de consultas** en base a métodos en interfaces
- Vamos a estudiarlo usando dos DBMS:
  1. Usando una base de datos en memoria H2
  2. Usando una base de datos persistente MySQL

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

- En las bases de datos en memoria (H2, Derby, HSQL…), el **esquema** se construye **automáticamente** al iniciar la aplicación
- Pasos para implementar una aplicación Spring Data JPA / Boot
  1. Configurar `pom.xml`
  2. Crear objetos de **entidad** (que serán mapeados en la base de datos)
  3. Crear **consultas** a la base de datos
  4. Hacer uso de base de datos
  5. Ejecutar la aplicación

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

1. Configurar `pom.xml`

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.2.7.RELEASE</version>
</parent>

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

<dependencies>
   <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

Java 8

Spring Data JPA

Base de datos H2

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos 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)

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

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

   - Vamos a crear consultas creando interfaces que extienden de la clase `CrudRepository`
   - El nombre de 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 los métodos:

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

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos 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/

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

3. Crear **consultas** a la base de datos
   - También podemos usar SQL directamente usando la anotación `@Query`

```
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();

}
```

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

4. Hacer uso de base de datos

```
@Component
public class DatabaseLoader {

    @Autowired
    private CustomerRepository repository;

    @PostConstruct
    private void initDatabase() {
        // Create
        repository.save(new Customer("John", "Doe"));
        repository.save(new Customer("Michael", "Smith"));
        // Update
        Customer firstCustomet = repository.findAll().iterator().next();
        System.out.println(firstCustomet);
        firstCustomet.setFirstName("Peter");
        repository.save(firstCustomet);
        // Read
        Iterable<Customer> all = repository.findAll();
        for (Customer customer : all) {
            System.out.println(customer);
        }
        // Delete
        long firstId = repository.findAll().iterator().next().getId();
        repository.delete(firstId);
        System.out.println(repository.count());
    }
}
```

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

Lo normal será hacer uso del repositorio en controladores MVC

2. Bases de datos relacionales con Spring

Spring Data JPA con base de datos H2

5. Ejecutar la aplicación

```
@SpringBootApplication
public class H2Application {

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

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.2.2.RELEASE)

...
Customer [id=1, firstName=John, lastName=Doe]
Customer [id=1, firstName=Peter, lastName=Doe]
Customer [id=2, firstName=Michael, lastName=Smith]
1
...
```

2. Bases de datos relacionales con Spring

Spring Data JPA con base persistente 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

2. Bases de datos relacionales con Spring

Spring Data JPA con base persistente MySQL

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

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.2.7.RELEASE</version>
</parent>

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

<dependencies>
   <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>
```

Spring Boot

Java 8

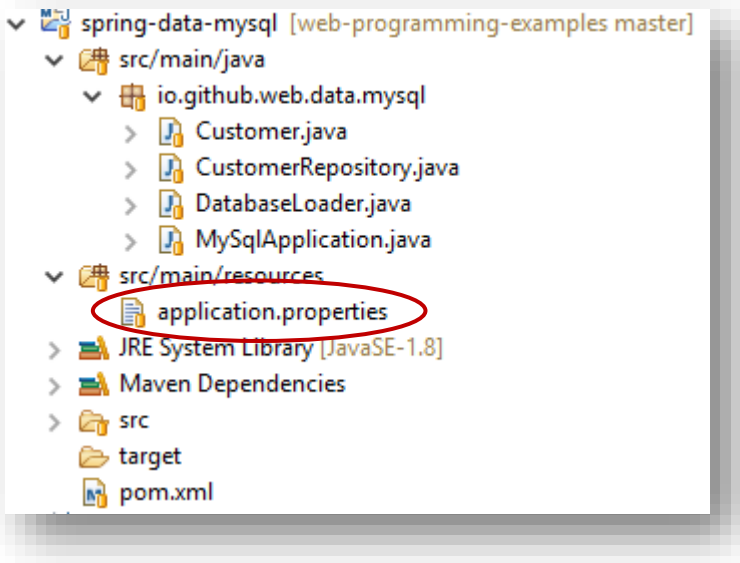
Spring Data JPA

MySQL

2. Bases de datos relacionales con Spring

Spring Data JPA con base persistente MySQL

- En segundo lugar hay que añadir un `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)

2. Bases de datos relacionales con Spring

Spring Data JPA con base persistente 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)

2. Bases de datos relacionales con Spring

Spring Data JPA con base persistente 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

Índice de contenidos

3. Bases de datos NoSQL

- El término NoSQL ("no sólo SQL") define una clase de DBMS que difieren del clásico modelo relacional:
  - No utilizan estructuras fijas como tablas para el almacenamiento de los datos
  - No usan el modelo entidad-relación
  - No suelen permitir operaciones JOIN (para evitar sobrecargas en búsquedas)
  - Arquitectura distribuida (los datos pueden estar compartidos en varias máquinas mediante mecanismos de tablas Hash distribuidas)
- Este tipo de bases de datos coincide con la explosión de usuarios que han experimentados algunas aplicaciones (por ejemplo Facebook, Twitter, YouTube, etc)

3. Bases de datos NoSQL

Ejemplo de uso de MongoDB en Java

- MondoDB driver:

```
<dependency>
   <groupId>org.mongodb</groupId>
   <artifactId>mongo-java-driver</artifactId>
   <version>3.0.0</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);
```

3. Bases de datos NoSQL

Ejemplo de uso de 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());
```

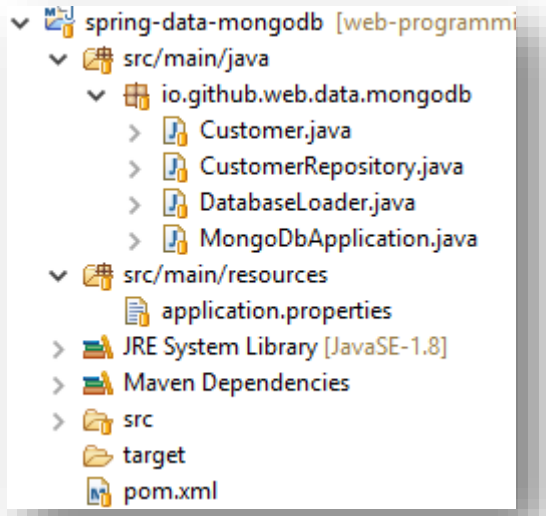
- Eliminar documento:

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

Índice de contenidos

4. Bases de datos NoSQL con Spring

Ejemplo de uso de MongoDB con Spring Boot



pom.xml

```xml
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.7.RELEASE</version>
</parent>

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

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

4. Bases de datos NoSQL con Spring

Ejemplo de uso de MongoDB con Spring Boot

application.properties

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

El identificador incremental en MongoDB es de tipo `String`, no entero

Customer.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
}
```