Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 2.4:

Spring Data

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

Introducción

- Dependencia diseñada para simplificar la implementación de capas de acceso a datos en aplicaciones Java.
- Reduce la cantidad de código boilerplate necesario para la implementación de capas de datos.
- Proporcionar una manera consistente y fácil de acceder a datos almacenados en diferentes tipos de bases de datos, tanto SQL como NoSQL.
- Fácil y rápida integración con Spring.

Introducción

- Esfuerzo reducido en el mantenimiento y actualización de las capas de acceso a datos.
- Facilidad de escalar y adaptarse a diferentes tecnologías de bases de datos.
- Permite a los desarrolladores enfocarse más en la lógica de negocio en lugar de las complejidades de la conectividad y operaciones de base de datos.

Introducción

- Spring Data JPA (Java Persistence API):
  - Proporciona integración con bases de datos relacionales mediante JPA.
  - Facilita la implementación de repositorios de datos para entidades JPA.
- Spring Data MongoDB:
  - Soporte específico para MongoDB, una base de datos NoSQL orientada a documentos.
  - Permite realizar operaciones de datos en documentos MongoDB con facilidad.
- Spring Data Redis:
  - Proporciona acceso y manejo de datos en Redis, una base de datos en memoria.
  - Ideal para casos de uso que requieren alta velocidad y escalabilidad.
- Spring Data Cassandra:
  - Soporte para Apache Cassandra, una base de datos NoSQL distribuida.
  - Permite manejar grandes cantidades de datos con alta disponibilidad.

Índice

- Introducción
- **JPA**
- Repositorios
- H2
- Tipos de correspondencia

JPA

- Spring Data JPA es un subproyecto de Spring Data que se centra en la integración con bases de datos relacionales usando la Java Persistence API (JPA).
- Facilita la implementación de capas de acceso a datos, proporcionando una manera más simple y concisa de realizar operaciones de base de datos.
- Proporciona interfaces de repositorio predefinidas, como CrudRepository y PagingAndSortingRepository.
- Simplifica las operaciones CRUD y de paginación/sorteo.

JPA

- Spring Data JPA es un subproyecto de Spring Data que se centra en la integración con bases de datos relacionales usando la Java Persistence API (JPA).
- Facilita la implementación de capas de acceso a datos, proporcionando una manera más simple y concisa de realizar operaciones de base de datos.
- Consta de varias características clave:

JPA

- Repositorio abstracción:
  - Proporciona interfaces de repositorio predefinidas, como CrudRepository y PagingAndSortingRepository.
  - Simplifica las operaciones CRUD y de paginación/sorteo.
- Mapeo Objeto-Relacional (ORM):
  - Integra JPA para mapear objetos en Java a tablas en bases de datos relacionales.
  - Automatiza la conversión entre registros de base de datos y objetos Java.
- Consultas derivadas:
  - Permite la creación de consultas a partir de los nombres de los métodos en las interfaces de repositorio.
  - Ejemplo: findByUsername(String username) automáticamente genera una consulta para buscar por el campo username.

JPA

- Annotaciones de consulta:
  - Utiliza la anotación @Query para definir consultas personalizadas utilizando JPQL (Java Persistence Query Language).
  - Permite escribir consultas complejas que no se pueden derivar directamente del nombre del método.
- Integración con Spring Framework:
  - Se integra sin problemas con otros componentes, como Spring Transaction Management.
  - Proporciona soporte para la gestión de transacciones y caché.
- Gestión de entidades:
  - Facilita la gestión del ciclo de vida de las entidades JPA, como persistencia, actualización y eliminación.
  - Soporta asociaciones entre entidades, como relaciones Uno-a-Uno, Uno-a-Muchos, y Muchos-a-Muchos.

Índice

- Introducción
- JPA
- **Repositorios**
- H2
- Tipos de correspondencia

Repositorios

- Un repositorio es un mediador entre el dominio de la aplicación y las operaciones de acceso a datos.
- Reducen la necesidad de código boilerplate y centralizan la lógica de acceso a datos.
- Tipos de Repositorios en Spring Data:
  - Diferentes tipos de interfaces de repositorio proporcionados por Spring Data, como CrudRepository, PagingAndSortingRepository, y JpaRepository.
- Operaciones CRUD:
  - Los repositorios simplifican las operaciones CRUD (Crear, Leer, Actualizar, Borrar).
  - Ejemplos de métodos comunes: save(), findAll(), findById(), y delete().

Repositorios

- CrudRepository es una de las interfaces de repositorio proporcionadas por Spring Data.
- Está diseñada para proporcionar operaciones CRUD (Crear, Leer, Actualizar, Borrar) para un tipo de entidad específico.
- Es una interfaz genérica que se parametriza con el tipo de la entidad y el tipo de su clave primaria.

UserRepository.java

```
import org.springframework.data.repository.CrudRepository;
public interface UserRepository extends CrudRepository<User, Long> {
}
```

Repositorios

- Crear y Actualizar:
  - save(S entity): Guarda una entidad dada. Si la entidad ya existe, la actualiza; de lo contrario, la crea.
  - saveAll(Iterable<S> entities): Guarda todas las entidades dadas.
- Leer:
  - findById(ID id): Recupera una entidad por su clave primaria.
  - findAll(): Devuelve todas las entidades.
  - findAllById(Iterable<ID> ids): Recupera todas las entidades cuyas claves primarias están en el iterable proporcionado.
- Borrar:
  - deleteById(ID id): Elimina la entidad con la clave primaria dada.
  - delete(S entity): Elimina la entidad dada.
  - deleteAll(): Elimina todas las entidades.

Índice

- Introducción
- JPA
- Repositorios
- **H2**
- Tipos de correspondencia

H2

- La base de datos H2 es un sistema de gestión de bases de datos relacionales escrito en Java. Es conocida por su pequeño tamaño, alta velocidad y por ser de código abierto.
- Base de Datos en Memoria y Basada en Disco:
  - H2 puede ser configurada tanto para almacenar sus datos en memoria como en disco.
  - Se usa a menudo para pruebas unitarias y de integración.
  - El modo basado en disco permite la persistencia de datos entre reinicios de la aplicación.
- Soporte JDBC y Compatibilidad SQL:
  - H2 proporciona un driver JDBC para conectar con la base de datos, permitiendo su uso con una gran variedad de herramientas y frameworks de Java.
  - Es compatible con la mayoría de las sintaxis y características de SQL, facilitando la migración desde otros sistemas de gestión de bases de datos SQL.

H2

- Fácil de Usar y Configurar:
  - H2 es muy simple de configurar y no requiere de una instalación compleja.
  - Puede ejecutarse como un proceso independiente o integrarse dentro de aplicaciones Java.
- Consola Web para la Gestión de la Base de Datos:
  - Incluye una consola web para facilitar la administración de la base de datos, permitiendo ejecutar consultas SQL, administrar tablas, entre otras tareas.
- Alto Rendimiento:
  - H2 está optimizada para un alto rendimiento, con velocidades de lectura y escritura muy rápidas, especialmente en el modo en memoria.

H2

- Soporte para Funcionalidades Avanzadas:
  - A pesar de su tamaño reducido, H2 soporta características avanzadas como transacciones multiversión (MVCC), cifrado de base de datos, y compresión de datos.
- Uso en Desarrollo y Pruebas:
  - Es comúnmente usada en entornos de desarrollo y pruebas debido a su facilidad de configuración y rápido despliegue.
  - Ideal para casos en los que se requiere una base de datos ligera y no se necesitan las capacidades completas de un sistema de gestión de bases de datos más grande.
- Soporte para Clientes de Múltiples Lenguajes:
  - Aunque H2 está escrita en Java, existen formas de interactuar con ella desde otros lenguajes de programación.

Ejemplo

Cerveza.java

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Cerveza {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String nombre;
    private String tipo;
    private Double alcohol; // Porcentaje de alcohol
}
```

- @Entity permite convertir una clase en Java en una tabla en la BD.
- @Id marca un determinado campo como clave primaria en la tabla que se genera en la BD.
- @GeneratedValue(…) incrementa el Id de forma automática cada vez que se añade un registro.
- El resto de los campos se convierten a los tipos correspondientes.

Ejemplo

pom.xml

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

- Añadimos la dependencia de la BD h2.
- Creamos como Interfaz el CervezaRepository

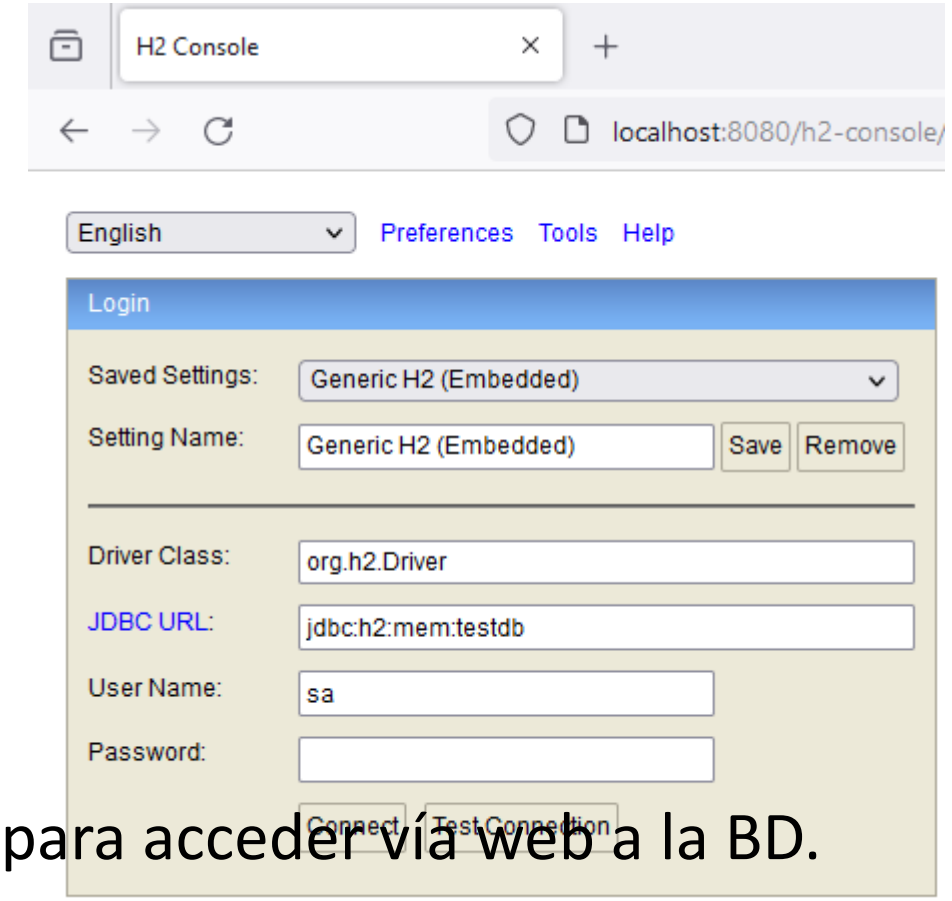
CervezaRepository.java

```
public interface CervezaRepository extends CrudRepository<Cerveza, Long> {
}
```

Ejemplo

application.properties

```
# Habilitar la consola H2
spring.h2.console.enabled=true

# (Opcional) Configurar el path de la consola H2
spring.h2.console.path=/h2-console

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```



- La primera línea nos permite habilitar la consola para acceder vía web a la BD.
- La segunda nos permite indicar la URL en la cual queremos que esté disponible la interfaz web.
- La tercera nos permite indicar donde está la BD.

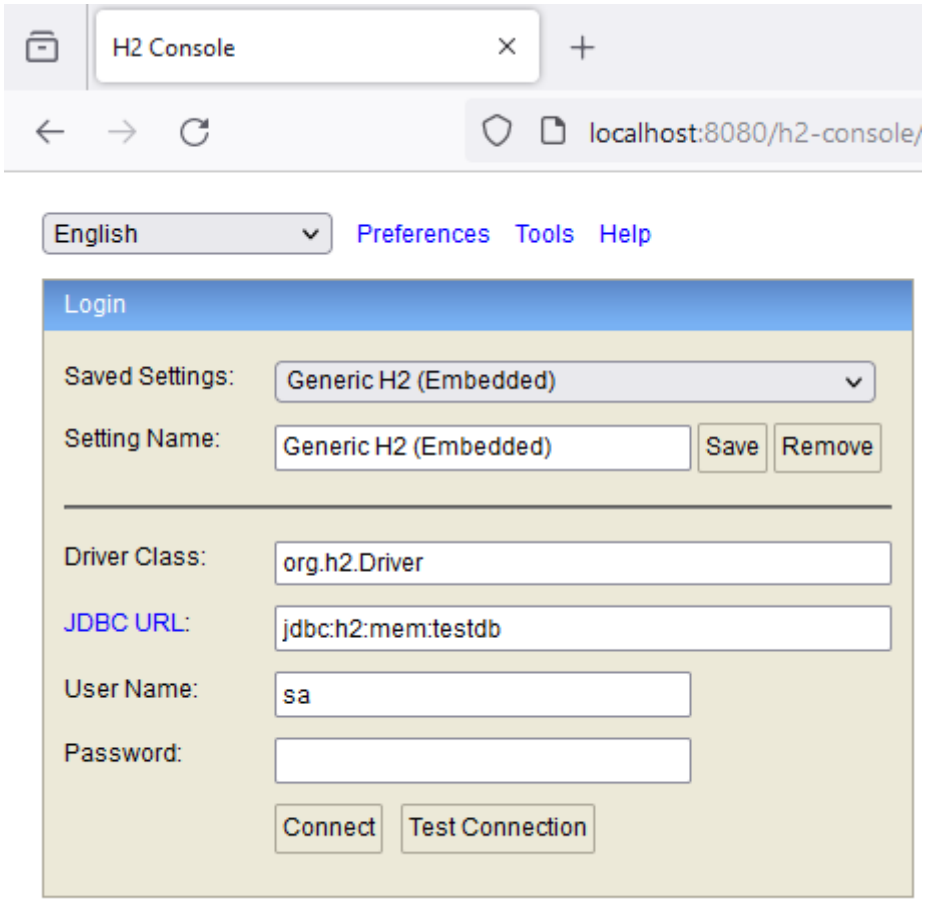
Ejemplo

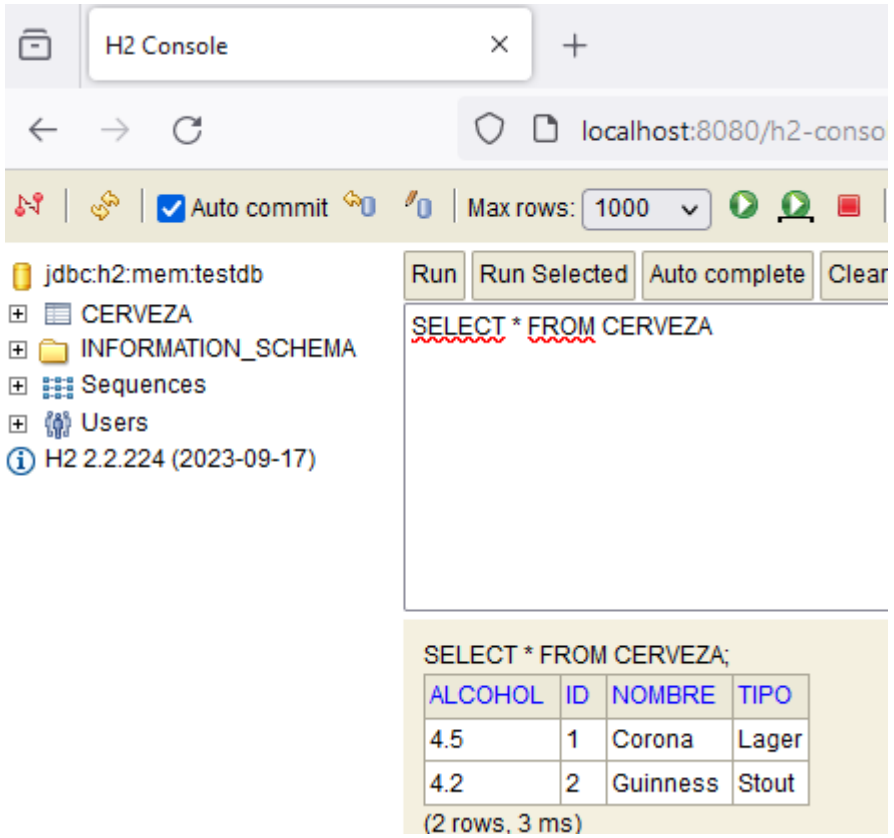
SpringDataEjemplo1Application.java

```
ConfigurableApplicationContext context =
SpringApplication.run(SpringDataEjemplo1Application.class, args);
CervezaRepository cervezaRepository = context.getBean(CervezaRepository.class);

// Añadir algunas cervezas
Cerveza cerveza1 = new Cerveza();
cerveza1.setNombre("Corona");
cerveza1.setTipo("Lager");
cerveza1.setAlcohol(4.5);
cervezaRepository.save(cerveza1);

Cerveza cerveza2 = new Cerveza();
cerveza2.setNombre("Guinness");
cerveza2.setTipo("Stout");
cerveza2.setAlcohol(4.2);
cervezaRepository.save(cerveza2);

// Leer y mostrar las cervezas
cervezaRepository.findAll().forEach(cerveza -> System.out.println(cerveza.getNombre() + " - " +
cerveza.getTipo() + " - " + cerveza.getAlcohol() + "%"));
```

Ejemplo

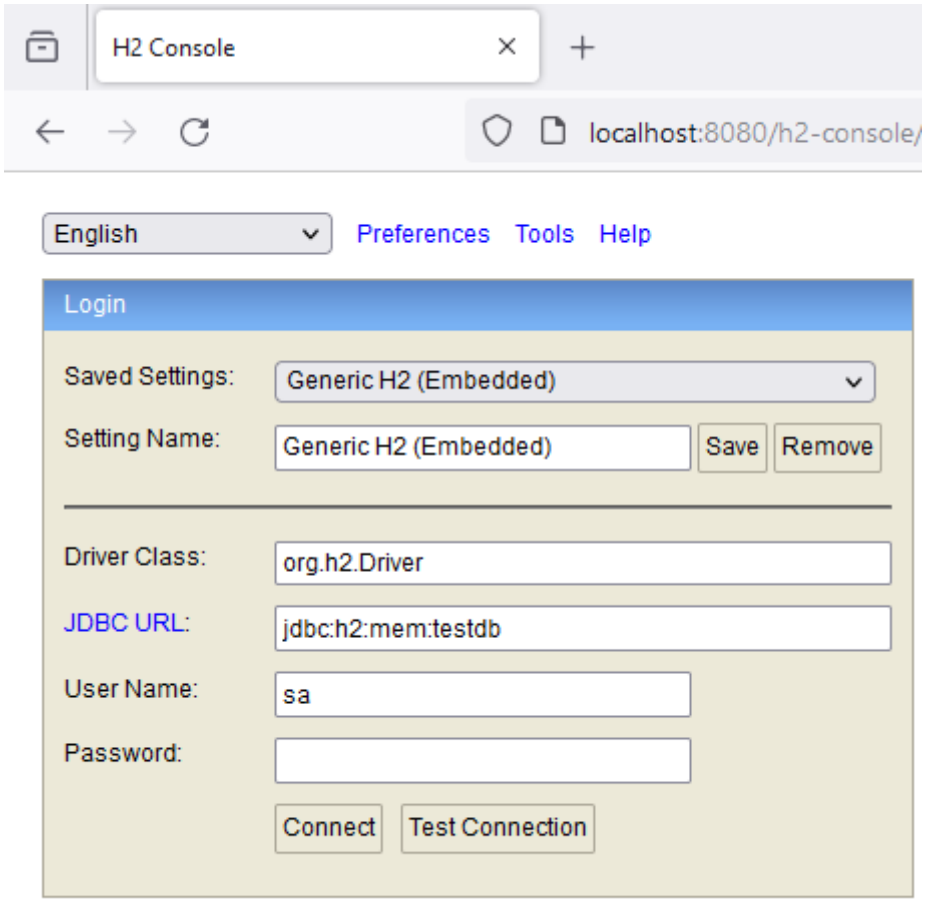
Resultado de la consulta

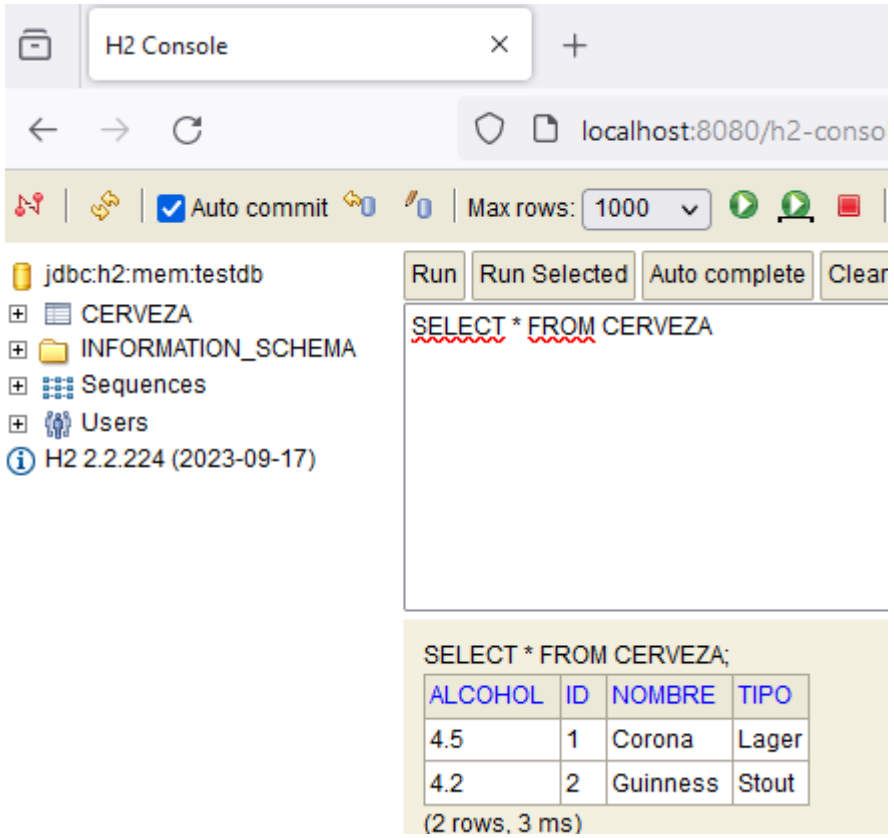




Ejemplo

Resultado de la consulta





Índice

Tipos de correspondencia

- La relación 1:N en JPA (Java Persistence API) es una de las relaciones más comunes.
- En este tipo de correspondencia una entidad (el "padre") puede tener múltiples instancias de otra entidad (los "hijos"), pero cada instancia hija está relacionada con una sola instancia padre.

Tipos de correspondencia

Autor.java

```
@Entity
public class Autor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    @OneToMany(mappedBy = "autor", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private Set<Libro> libros = new HashSet<>();
    // Constructor, Getters y Setters
}
```

- Se añade @OneToMany para indicar en qué dirección se propaga la clave ajena.
- Se indica en autor el nombre del campo en la otra tabla.
- Se indica que el borrado será en cascada.

Tipos de correspondencia

Libro.java

```
@Entity
public class Libro {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String titulo;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "autor_id")
    private Autor autor;
    // Constructor, Getters y Setters
}
```

- Se añade @ManyToOne para indicar en qué dirección viene la clave ajena.
- En términos generales, suele ser de ayuda para el programador.

Tipos de correspondencia

LibroRepository.java

```
public interface LibroRepository extends JpaRepository<Libro, Long> {
}
```

AutorRepository.java

```
public interface AutorRepository extends JpaRepository<Autor, Long> {
}
```

- Se añaden los repositorios de las entidades.
- Se puede usar el CrudRepository también.

Tipos de correspondencia

DataLoader.java

```java
@Component
public class DataLoader implements CommandLineRunner
{
    @Autowired
    private AutorRepository autorRepository;

    @Autowired
    private LibroRepository libroRepository;

    @Override
    public void run(String... args) throws Exception {
        Autor autor = new Autor();
        autor.setNombre("Gabriel García Márquez");

        Libro libro = new Libro();
        libro.setTitulo("Cien años de soledad");
        libro.setAutor(autor);
        autor.getLibros().add(libro);

        autorRepository.save(autor);
    }
}
```

- DataLoader, que implementa CommandLineRunner, se usa para poblar la base de datos con un autor y un libro al iniciar la aplicación.
- Se añaden con el @Autowired ambos repositorios.
- El libroRepository es de utilidad para operaciones directas sobre libros.
- **Es necesario configurar H2 en el application.properties.**
- Una buena idea es revisar a través de la interfaz web el contenido de las tablas.

Tipos de correspondencia

- El tipo de correspondencia N:M en JPA (Java Persistence API) permite que múltiples instancias de una entidad estén asociadas con múltiples instancias de otra entidad.
- Este tipo de relación se gestiona mediante una tabla de unión que mantiene las referencias cruzadas entre las dos entidades.
- Consideremos un ejemplo clásico: la relación entre Estudiante y Curso, donde un estudiante puede inscribirse en varios cursos y cada curso puede tener varios estudiantes.

Tipos de correspondencia

Estudiante.java

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Estudiante {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;

    @ManyToMany(mappedBy = "estudiantes")
    private Set<Curso> cursos = new HashSet<>();
}
```

pom.xml

```xml
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.20</version>
  <scope>provided</scope>
</dependency>
```

- Lombok es una dependencia que permite evitar el código repetido a través de anotaciones.

Tipos de correspondencia

Curso.java

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Curso {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String titulo;

    @ManyToMany
    @JoinTable(
            name = "curso_estudiante",
            joinColumns = @JoinColumn(name = "curso_id"),
            inverseJoinColumns = @JoinColumn(name = "estudiante_id")
    )
    private Set<Estudiante> estudiantes = new HashSet<>();
}
```

- @ManyToMany permite indicar el nombre la tabla que se genera a partir del tipo de correspondencia N:M.
- Además, es necesario indicar la clave primaria. Esta clave primaria está compuesta por dos claves ajenas. Una de estas claves sale de esta misma tabla, que se indica en el joinColumns. La otra parte de la clave viene de la otra tabla y se indica como inverseJoinColumns.
- No se recomienda el uso de @Data en este tipo de correspondencia.

Tipos de correspondencia

DataLoader.java

```java
@Component
public class DataLoader implements CommandLineRunner {
    @Autowired
    private CursoRepository cursoRepository;
    @Autowired
    private EstudianteRepository estudianteRepository;
    @Override
    public void run(String... args) throws Exception {
        // Crear estudiantes
        Estudiante estudiante1 = new Estudiante();
        estudiante1.setNombre("Ana");
        Estudiante estudiante2 = new Estudiante();
        estudiante2.setNombre("Luis");
        // Crear cursos
        Curso curso1 = new Curso();
        curso1.setTitulo("Matemáticas");
        Curso curso2 = new Curso();
        curso2.setTitulo("Literatura");
        // Asignar estudiantes a cursos
        curso1.getEstudiantes().add(estudiante1);
        curso1.getEstudiantes().add(estudiante2);
        curso2.getEstudiantes().add(estudiante1);
        estudiante1.getCursos().add(curso1);
        estudiante1.getCursos().add(curso2);
        estudiante2.getCursos().add(curso1);
        // Guardar en la base de datos
        estudianteRepository.save(estudiante1);
        estudianteRepository.save(estudiante2);
        cursoRepository.save(curso1);
        cursoRepository.save(curso2);
    }
}
```

- Es necesario implementar los repositorios correspondientes a cada entidad.
- Es necesario modificar el application.properties con la configuración de H2.
- Utilizamos el DataLoader para cargar los datos del ejemplo.
- A continuación, se puede utilizar la interfaz web con H2 para comprobar el correcto funcionamiento.

Ejercicio

- Para generar un modelo relacional que incluya la entidad Cerveza y tenga relaciones de tipo (1:N) y (N:M), podemos introducir dos entidades adicionales: Cervecero (para representar a un productor de cerveza) e Ingrediente.
- El diagrama de este modelo mostraría tres entidades (Cervecero, Cerveza, Ingrediente) con las relaciones correspondientes. La relación 1:N entre Cervecero y Cerveza se maneja mediante una clave foránea en Cerveza. La relación N:M entre Cerveza e Ingrediente se gestiona a través de una tabla de unión que conecta las claves primarias de ambas entidades.

Ejercicio – Entidades y relaciones

- Entidad Cervecero:
  - Representa a los productores de cerveza.
  - Cada Cervecero puede producir varias Cervezas, pero cada Cerveza es producida por un único Cervecero. Esto establece una relación (1:N) entre Cervecero y Cerveza.
- Entidad Cerveza:
  - Ya definida, con atributos como id, nombre, tipo, y alcohol.
- Entidad Ingrediente:
  - Representa los ingredientes utilizados en la elaboración de cervezas.
  - Una Cerveza puede contener varios Ingredientes, y un Ingrediente puede ser utilizado en varias Cervezas. Esto establece una relación "Muchos a Muchos" (N:M) entre Cerveza y Ingrediente.

Ejercicio – Modelo relacional

- Cervecero
  - id: Clave primaria.
  - nombre: Nombre del cervecero o de la empresa cervecera.
- Cerveza
  - (Los campos de siempre)
  - cervecero_id: Clave foránea que referencia a Cervecero.
- Ingrediente
  - id: Clave primaria.
  - nombre: Nombre del ingrediente (por ejemplo, lúpulo, malta, levadura, etc.).
- Cerveza_Ingrediente (Tabla de Unión para la relación N:M)
  - cerveza_id: Clave foránea que referencia a Cerveza.
  - ingrediente_id: Clave foránea que referencia a Ingrediente.

Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 2.4:

Spring Data

Nicolás Rodríguez
nicolas.rodriguez@urjc.es