Grupo 29

Item 4. Implementación de una aplicación Full-Text Search

Manual de instrucciones

Khawla Al Asfar

Juan Carlos Utrilla Martín

Juan Rodríguez Dueñas

Yassine Taziny

José Manuel Lara Morilla

Contenido

1. Insertar dependencias necesarias

Para comenzar, deberemos de añadir las dependencias de las que se van a hacer uso para implementar una aplicación que use una búsqueda en la base de datos a través de *Full-Text Search*. Para ello, debemos de dirigirnos a nuestro archivo **pom.xml** de nuestro proyecto Maven e importar las siguientes dependencias: *hibernate-search* e *hibernate-search-analizers*.

```
<!-- Full-Text Search -->

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-search</artifactId>
    <version>4.5.0.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-search-analyzers</artifactId>
    <version>4.5.0.Final</version>
</dependency>
```

Acto seguido, deberemos de actualizar el proyecto Maven para que descargue e instale dichas dependencias.

2. Trip.java

Como nos piden que realicemos una búsqueda de *trips* a través de una *keyword* que realizará una búsqueda en los atributos *ticker*, *title* y *description*, se procederá a añadir una serie de anotaciones necesarias para crear la arquitectura de búsqueda de la aplicación.

```
@Entity
@Access(AccessType.PROPERTY)
@Indexed
public class Trip extends DomainEntity {
```

En la cabecera de la clase deberemos de añadir la anotación *@Indexed*, la cual indica a *Hobernate Search* que *Trip* será una entidad que necesita ser indexada para la búsqueda.

```java
@Column(unique = true)
@Field(index = Index.YES, analyze = Analyze.YES, store = Store.NO)
@NotBlank
@Pattern(regexp = "^(\\d{2})(\\d{2})(\\d{2})\\-([A-Z]{4})$")
public String getTicker() {
    return this.ticker;
}

public void setTicker(final String ticker) {
    this.ticker = ticker;
}

@Field(index = Index.YES, analyze = Analyze.YES, store = Store.NO)
@NotBlank
public String getTitle() {
    return this.title;
}

public void setTitle(final String title) {
    this.title = title;
}

@Field(index = Index.YES, analyze = Analyze.YES, store = Store.NO)
@NotBlank
public String getDescription() {
    return this.description;
}

public void setDescription(final String description) {
    this.description = description;
}
```

Debemos de añadir la anotación *@Field* junto con los parámetros *index*, *analize* y *store* para especificar a *Hibernate Search* como debe actuar con cada variable que va a repercutir en la búsqueda:

- *index = Index.YES* → Indica que la variable será indexada.
- *analize = Analize.YES* → Indica que se realice un análisis previo de optimización para mejorar la búsqueda de los tipos de datos.
- *store = Store.NO* → Indica que no se almacene el valor en el índice.

3. FullTextApplication.java

Esta clase se localizará en *src/main/java/utilities* y será la que implemente toda la funcionalidad de nuestra aplicación.

```
public class FullTextApplication {

    public static void main(final String[] args) throws Throwable {

        final DatabaseUtil du = new DatabaseUtil();
        du.initialise();
        final FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(du.getEntityManager());

        FullTextApplication.indexTrips(du, fullTextEntityManager);
        final ConsoleReader cr = new ConsoleReader();
        String line = null;

        System.out.println("\n***** Full-Text Application *****\n\n");
        System.out.println("Enter keyword to find Trip by ticker, title or description: \n");

        do
            try {
                line = cr.readCommand();
                final List<Trip> trips = FullTextApplication.keywordSearch(line, fullTextEntityManager);
                if (trips.isEmpty())
                    System.out.println("There aren't coincidences with the keyword inserted.\n");
                else
                    SchemaPrinter.print(trips);
            } catch (final Throwable oops) {

            }
        while (true);
    }
```

El método *main* describirá como se comportará la aplicación y que operaciones debe de realizar en cada caso.

En la primera parte se inicializara la base de datos a través de *du.initialise()*, que usaremos para obtener la variable *fullTextEntityManager*, que se usará para reconstruir el índice y realizar la búsqueda.

En la segunda se ejecutará el método *indexTrips*, el cual reconstruirá el índice para la entidad *Trip*. También se inicializará una variable de tipo *ConsoleReader*, la cual nos permitirá leer de la consola el *keyword* introducido por el usuario.

Dentro de la implementación también llamaremos a la clase *SchemaPrinter*, donde encontraremos el método *print* que formateará cada *trip* devuelto por el método *keywordSearch()*.

```
public static void indexTrips(final DatabaseUtil du, final FullTextEntityManager fullTextEntityManager) throws Exception {

    try {
        du.getEntityManager().getTransaction().begin();
        fullTextEntityManager.createIndexer().startAndWait();
    } catch (final Exception e) {
        throw e;
    }
}
```

El método *indexTrips* realizará la reconstrucción del índice de la entidad *Trip* pasándole como parámetros de tipo *DatabaseUtil* (previamente inicializada) y *FullTextEntityManager*.

```
@SuppressWarnings("unchecked")
public static List<Trip> keywordSearch(final String keywordSearch, final FullTextEntityManager fullTextEntityManager) throws InstantiationException, IllegalAccessException, ClassNotFoundException {

    final QueryBuilder qb = fullTextEntityManager.getSearchFactory().buildQueryBuilder().forEntity(Trip.class).get();
    final org.apache.lucene.search.Query query = qb.keyword().onFields("ticker", "title", "description").matching(keywordSearch).createQuery();
    final Query fullSearchQuery = fullTextEntityManager.createFullTextQuery(query, Trip.class);
    final List<Trip> result = fullSearchQuery.getResultList();
    return result;
}
```

El método *keywordSearch* implementará el código para la búsqueda de los *trips* dentro de la base de datos pasándole como parámetro el propio *keyword* y el *fullTextEntityManager*.

La anotación *@SuppressWarnings("unchecked")* para evitar una alerta de Eclipse en el método *getResultList()*.

4. DatabaseUtil.java

```
public EntityManager getEntityManager() {
    return this.entityManager;
}
```

Solamente realizar una pequeña mención a esta clase debido a que se ha de añadir un método *getEntityManager()* que devuelva la variable *entityManager* de la clase, la cual será necesaria para obtener la variable previamente nombrada como *fullTextEntityManager* y para comezar la transacción de la base de datos dentro del método *indexTrips()*.

5. Probando la aplicación

```
2017-10-30 18:56:07,585 [main] WARN  org.hibernate.search.impl.ConfigContext  - HSEARCH000075: Configuration setting hib
ernate.search.lucene_version was not specified, using LUCENE_CURRENT.
2017-10-30 18:56:07,585 [main] WARN  org.hibernate.search.impl.ConfigContext  - HSEARCH000075: Configuration setting hib
ernate.search.lucene_version was not specified, using LUCENE_CURRENT.

***** Full-Text Application *****


Enter keyword to find Trip by ticker, title or description:

> AAAA;
domain.Trip{id=7151, version=0}
        domain.DomainEntity::id: int = 7151
        domain.DomainEntity::version: int = 0
        domain.Trip::ticker: java.lang.String = "170101-AAAA"
        domain.Trip::title: java.lang.String = "Trip 1"
        domain.Trip::description: java.lang.String = "This is trip description"
        domain.Trip::requirements: java.util.Collection = ["Requirement 1"]
        domain.Trip::publicationDate: java.util.Date = <<2017-02-01 12:00:00.0>>
        domain.Trip::startDateTrip: java.util.Date = <<2017-03-01 12:00:00.0>>
        domain.Trip::endDateTrip: java.util.Date = <<2017-04-01 12:00:00.0>>
        domain.Trip::cancelledReason: java.lang.String = null
        domain.Trip::price: domain.Money = domain.Money@4802c1ef
        domain.Trip::stages: java.util.Collection = [domain.Stage{id=7045, version=0}, domain.Stage{id=7046, version=0}, domain
.Stage{id=7047, version=0}, domain.Stage{id=7048, version=0}, domain.Stage{id=7049, version=0}]
        domain.Trip::category: domain.Category = domain.Category{id=7032, version=0}
        domain.Trip::registers: java.util.Collection = [domain.Register{id=7202, version=0}]
        domain.Trip::legalText: domain.LegalText = domain.LegalText{id=7075, version=0}
        domain.Trip::notes: java.util.Collection = [domain.Note{id=7161, version=0}, domain.Note{id=7170, version=0}]
        domain.Trip::auditRecords: java.util.Collection = [domain.AuditRecord{id=7173, version=0}]
        domain.Trip::sponsorships: java.util.Collection = [domain.Sponsorship{id=7180, version=0}]
        domain.Trip::survivalClasses: java.util.Collection = [domain.SurvivalClass{id=7182, version=0}]
        domain.Trip::manager: domain.Manager = domain.Manager{id=7024, version=0}
        domain.Trip::ranger: domain.Ranger = domain.Ranger{id=7026, version=1}
        domain.Trip::stories: java.util.Collection = [domain.Story{id=7192, version=0}]
        domain.Trip::finders: java.util.Collection = [domain.Finder{id=7086, version=0}, domain.Finder{id=7087, version=0}]
        domain.Trip::applications: java.util.Collection = [domain.Application{id=7224, version=0}, domain.Application{id=7234,
version=0}]
> ABCD;
There aren't coincidences with the keyword inserted.

>
```

Para iniciar la aplicación solamente deberemos de ejecutar el archivo *FullTextApplication.java* como aplicación Java. La aplicación inicializará la base de datos y mostrará un texto el cual pedirá al usuario introducir un *keyword*. Como vemos en la imagen, hemos introducido el *keyword* AAAA junto con un ";". Al introducir dicha *keyword*, la aplicación nos devolverá el *trip1*, cuyo *ticker* es "170101-AAAA". En el caso de introducir un *keyword* que no coincida con ningún atributo de la entidad *Trip*, la aplicación nos devolverá un mensaje diciendo que no se han encontrado coincidencias con el *keyword* insertado.

6. Referencias

- **Hibernate Search and JPA tutorial**: http://www.mastertheboss.com/jboss-frameworks/hibernate-jpa/hibernate-search/hibernate-search-and-jpa-tutorial
- **Integrating Full Text Search to Spring MVC with Hibernate**: https://www.codeproject.com/Articles/830529/Integrating-Full-Text-Search-to-Spring-MVC-with-Hi
- **Getting started with Hibernate Search**: http://hibernate.org/search/documentation/getting-started/