REPORT A+

Deliverable D03 – Persistence Models

Antonio Nolé Anguita
María Jiménez Vega
Álvaro Calle González
Julia García Gallego
Fernando Manuel Ruiz Pliego

Index

Introduction

The application uses Hibernate to integrate Apache Lucene in our project, which is a component that allows us to implement Full-Text queries. This kind of queries give us the possibility to search for objects that contain a key word in a string attribute, in a more efficient way than using queries that include the operator 'LIKE'.

Content

In order to implement this application, we have followed the next steps:

1. Choosing a compatible version of Hibernate Search

Hibernate Search is the component that integrate Apache Lucene in our project. We have to search here for the compatible version with our current environment.

In our case, the latest compatible version is Hibernate Search 5.3.0.

2. Adding Hibernate Search to Maven dependencies

Now we have to add the version of Hibernate Search chosen to pom.xml, which is the file that manages Maven dependencies.

In our case:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-search-orm</artifactId>
    <version>5.3.0.Final</version>
</dependency>
```

3. Hibernate Search configuration

a. Persistence.xml

We need to add some properties to persistence.xml. The first two lines tells Hibernate Search where to store the index files of Apache Lucene. In our case we store it in Acme-HandyWorker/var/lucene/indexes.

The last line is not a mandatory property but recommended one. The purpose of this line is to make it easy to port the application to newer versions of Apache Lucene. In the case of upgrading to a newer version, this line instructs some classes to conform to their behavior as defined in an (older) specific version of Lucene.

```
<property name="hibernate.search.default.directory_provider" value="filesystem"/>
<property name="hibernate.search.default.indexBase" value="var/lucene/indexes"/>
<property name="hibernate.search.lucene_version" value="LUCENE_53"/>
```

b. Java annotations

Now we need to mark which classes and attributes Apache Lucene must to index. This can be done using some annotations. In our case, full-text queries only involve our domain entity *FixUpTask* and its attributes *Ticker*, *Description* and *Address*, so we will use only some annotations. There are many others which still are very useful, but they are not necessary in our case. (See this for more information).

These are the annotations that we used:

- @Indexed: Marks our domain entity as indexable
- @Field: You have to mark the fields you want to make searchable of the class marked with @Indexed

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

    ...

    // Atributes

    private String  ticker;
    private Date    publicationMoment;
    private String  description;
    private String  address;
    private double  maxPrice;
    private Date    startDate;
    private Date    endDate;
```

```
    @Pattern(regexp = "\\d{6}-[A-Z0-9]{6}")
    @NotBlank
    @Column(unique = true)
    @Field
    public String getTicker() {
        return this.ticker;
    }

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

    ...

    @NotBlank
    @Field
    public String getDescription() {
        return this.description;
    }

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

    @NotBlank
    @Field
    public String getAddress() {
        return this.address;
    }

    public void setAddress(final String address) {
        this.address = address;
    }

    ...

}
```

4. Coding the application

We already have configured hibernate search. Now we will start to code our application.

a. Indexing

The first step is to create a routine that create initial indexes for the existing data in our database. After that, Hibernate Search will transparently index every entity persisted, updated or removed. You can achieve this by using the following code snippet:

```
EntityManager em = entityManagerFactory.createEntityManager();
FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(em);
fullTextEntityManager.createIndexer().startAndWait();
```

In our case we added a method on *DatabaseUtil.java* that returns *FullTextEntityManager* object:

DatabaseUtil.java:

```
public FullTextEntityManager getFullTextEntityManager() {
    return org.hibernate.search.jpa.Search
              .getFullTextEntityManager(this.entityManager);
}
```

LuceneUtil.java:

```
public static void initialiseIndex(final DatabaseUtil databaseUtil)
                  throws Throwable {
    FullTextEntityManager fullTextEntityManager;

    fullTextEntityManager = databaseUtil.getFullTextEntityManager();
    fullTextEntityManager.createIndexer().startAndWait();
}
```

b. Searching

The following step is making a routine that build the desired query. You can do this following this code snippet:

LuceneUtil.java:

```
public static List<?> findFixUpTaskByKeyword(final FullTextEntityManager
                                  fullTextEntityManager, final String keyword) {
    QueryBuilder qb;
    org.apache.lucene.search.Query query;
    javax.persistence.Query persistenceQuery;

    qb = fullTextEntityManager.getSearchFactory().buildQueryBuilder()
                              .forEntity(FixUpTask.class).get();
    query = qb.keyword().onFields("ticker",
                                  "description",
                                  "address").matching(keyword).createQuery();
    persistenceQuery = fullTextEntityManager
                                  .createFullTextQuery(query, FixUpTask.class);

    return persistenceQuery.getResultList();
}
```

c. Making the application interactive through console

The final step we have to do is to join these routines. We will explain the code of *FullTextSearchFixUpTask.java* so to follow this explanation, please see Acme-HandyWorker/src/main/java/utilities/aplus/d03/FullTextSearchFixUpTask.java.

First of all, we need to initialize the index calling the previous routine explained.

Then we need to start a transaction before we execute the full-text query. We call the routine that creates the query passing it the keyword introduced through the console (using *ConsoleReader.java*). Finally, we print the results (using *SchemaPrinter.java*) and commit the transaction.

In our application the transaction is in a loop that breaks when you don't introduce any key word to the console and then press enter.

And finally, we close the persistence context.

Bibliography

- https://docs.jboss.org/hibernate/search/5.3/reference/en-US/html_single/
- http://hibernate.org/search/documentation/