

ENTREGA 3

Report item 4

Diseño y Pruebas



Grupo 20:

José Ángel Domínguez Espinaco

Daniel Lozano Portillo

José Joaquín Rodríguez Pérez

María Ruiz Gutiérrez

Miguel Ternero Algarín

Laura Vera Recacha

Índice:

Introducción	3
Implementacion.....	3
Resultados	7
Bibliografia:	8

Introducción

En este documento se explica detalladamente como se llevó a cabo la implementación de la utilidad Hibernate Search con JPA. También se muestran los resultados y algunas partes del código implementado para su funcionamiento adecuado.

Implementación

Lo primero que hay que hacer es añadir la dependencia en el archivo pom.xml

```
<dependency>

  <groupId>org.hibernate</groupId>

  <artifactId>hibernate-search-orm</artifactId>

  <version>5.8.1.Final</version>

</dependency>
```

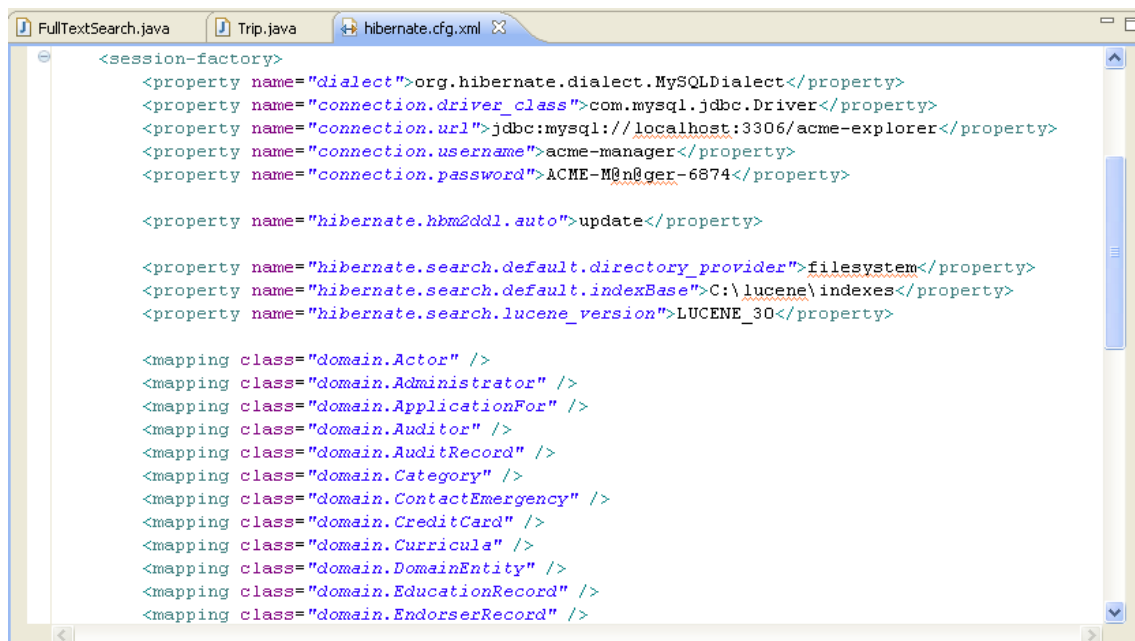
Si está utilizando Hibernate a través de JPA, se agregó estas mismas propiedades a persistence.xml. El siguiente paso es crear el archivo hibernate.cfg.xml en *src/MAIN/resources* y agregar tanto las clases de su dominio como la referencia a la base de datos.

```
<property name="hibernate.search.default.directory_provider"

  value="filesystem"/>

<property name="hibernate.search.default.indexBase"

  value="/var/lucene/indexes"/>
```



```

<session-factory>
  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="connection.url">jdbc:mysql://localhost:3306/acme-explorer</property>
  <property name="connection.username">acme-manager</property>
  <property name="connection.password">ACME-M@n@ger-6874</property>

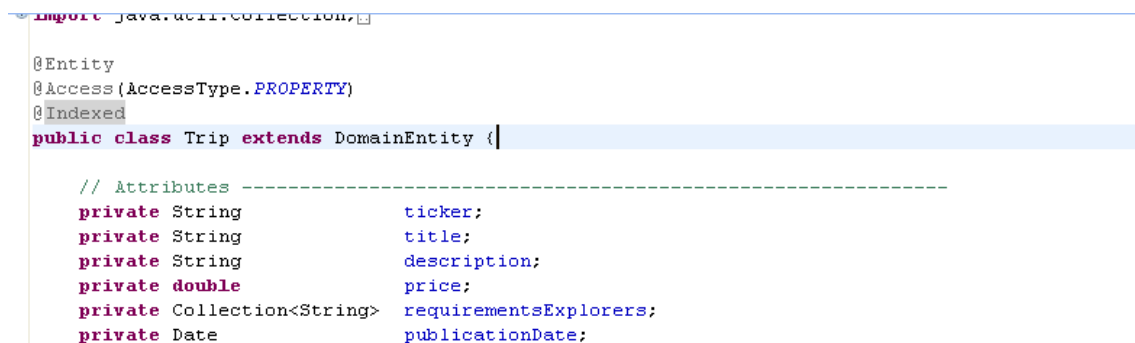
  <property name="hibernate.hbm2ddl.auto">update</property>

  <property name="hibernate.search.default.directory_provider">filesystem</property>
  <property name="hibernate.search.default.indexBase">C:\lucene\indexes</property>
  <property name="hibernate.search.lucene_version">LUCENE_30</property>

  <mapping class="domain.Actor" />
  <mapping class="domain.Administrator" />
  <mapping class="domain.ApplicationFor" />
  <mapping class="domain.Auditor" />
  <mapping class="domain.AuditRecord" />
  <mapping class="domain.Category" />
  <mapping class="domain.ContactEmergency" />
  <mapping class="domain.CreditCard" />
  <mapping class="domain.Curricula" />
  <mapping class="domain.DomainEntity" />
  <mapping class="domain.EducationRecord" />
  <mapping class="domain.EndorserRecord" />

```

Una vez realizado esto añadimos a la clase Trip @Indexed ya que es la clase que se va Indexar.



```

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

    // Attributes -----
    private String      ticker;
    private String      title;
    private String      description;
    private double      price;
    private Collection<String> requirementsExplorers;
    private Date        publicationDate;

```

A continuación, debe añadir la anotación @Field en los atributos que desee que sean buscados. En nuestro caso serán ticker, title, y description.

@Field(index = Index.YES, analyze = Analyze.NO, store = Store.YES)

El índice de parámetro = Index.YES asegurará que el texto será indexado, mientras que analyse = Analyze.YES asegura que el texto será analizado usando el analizador Lucene predeterminado. El tercer parámetro, store = Store.NO, garantiza que los datos reales no se almacenarán en el índice. Que estos datos estén almacenados en el índice o no, no tiene

nada que ver con la capacidad de buscarlos: el beneficio de almacenarlos es la capacidad de recuperarlos mediante proyecciones

```
@NotBlank
@Column(unique = true)
@Pattern(regexp = "[0-9]{2}[0-1-9]{1}|1[0-2]{1})((0|1|2)[0-9]{1}|3[0-1]{1})\\-[A-Z]{4}$")
@Field(index = Index.YES, analyze = Analyze.YES, store = Store.NO)
public String getTicker() {
    return this.ticker;
}

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

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

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

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

Ya solo nos faltaría crear la Clase para que todo esto funcione.

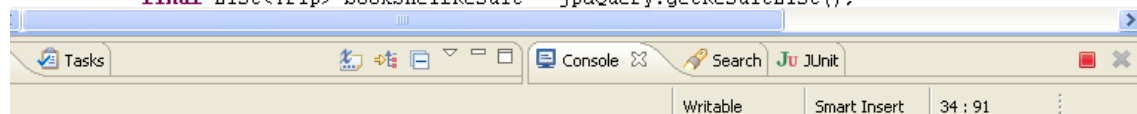
```
public static void main(final String[] args) {
    EntityManager entityManager = Persistence.createEntityManagerFactory("Acme-Explorer").cr
    FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(entityMana
    try {
        // This will ensure that index for already inserted data is created.
        fullTextEntityManager.createIndexer().startAndWait();
        // Search for Trip
        QueryBuilder qb = fullTextEntityManager.getSearchFactory().buildQueryBuilder().forEn
        org.apache.lucene.search.Query query = qb.keyword().onFields("ticker", "title", "des

        Query jpaQuery = fullTextEntityManager.createFullTextQuery(query, Trip.class);
        List<Trip> tripResult = jpaQuery.getResultList();

        for (Trip t : tripResult)
            System.out.println("Trip " + t.getTicker() + ", title: " + t.getTitle() + ", de

        if (tripResult != null)
            for (Trip TripEntityBean : tripResult)
                System.out.println("Trip found = " + TripEntityBean);
        // Search for book shelf
        qb = fullTextEntityManager.getSearchFactory().buildQueryBuilder().forEntity(Trip.cla
        query = qb.keyword().onFields("ticker").matching("Technical").createQuery();
        jpaQuery = fullTextEntityManager.createFullTextQuery(query, Trip.class);

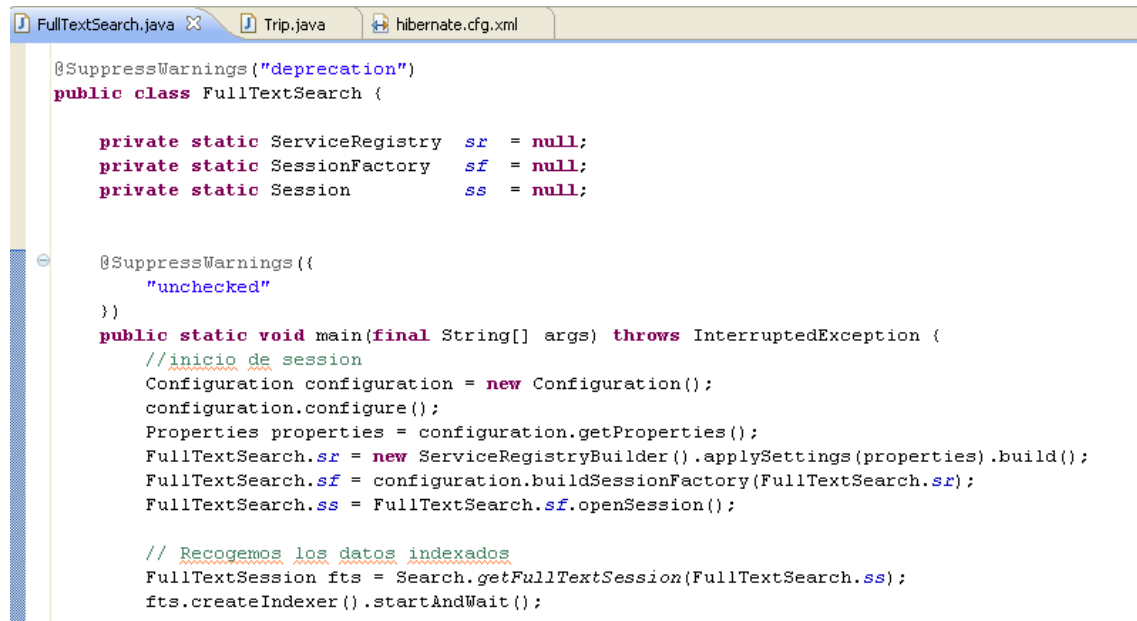
        // execute search
        final List<Trip> bookShelfResult = jpaQuery.getResultList();
    }
}
```



Cuando ya estaba todo listo para funcionar, al ejecutar hibernate, mostro un error de compatibilidad con la versión, por lo que procedimos a hacerlo con la versión 4.5.0 que encontramos que era compatible con nuestra versión de JPA e hibernate.

Finalmente, la versión más nueva no usaba la misma forma para indexar los datos ya populados en la base de datos con lo que se tuvo que cambiar. Y este fue el resultado:

En esta primera imagen se muestra el inicio de sesión y como se recogen los datos indexados.



```
@SuppressWarnings("deprecation")
public class FullTextSearch {

    private static ServiceRegistry sr = null;
    private static SessionFactory sf = null;
    private static Session ss = null;

    @SuppressWarnings({
        "unchecked"
    })
    public static void main(final String[] args) throws InterruptedException {
        //inicio de session
        Configuration configuration = new Configuration();
        configuration.configure();
        Properties properties = configuration.getProperties();
        FullTextSearch.sr = new ServiceRegistryBuilder().applySettings(properties).build();
        FullTextSearch.sf = configuration.buildSessionFactory(FullTextSearch.sr);
        FullTextSearch.ss = FullTextSearch.sf.openSession();

        // Recogemos los datos indexados
        FullTextSession fts = Search.getFullTextSession(FullTextSearch.ss);
        fts.createIndexer().startAndWait();
    }
}
```

En la siguiente como se muestran los datos de las Trip indexadas y se pide por pantalla una palabra para la búsqueda y con lucene se busca mediante una query de lucene.

```
//mostramos los datos
List<Trip> tripResult = FullTextSearch.ss.createQuery("from Trip").list();

for (final Trip t : tripResult)
    System.out.println("Trip " + t.getTicker() + ", title: " + t.getTitle() + ", descri

Scanner scn = new Scanner(System.in);
String tecladoString = null;

System.out.println("\nEscriba la palabra clave por la que quiera buscar: ");

//palabra que vamos a escribir por pantalla
tecladoString = scn.nextLine();

// lucene prepara la query
QueryBuilder qb = fts.getSearchFactory().buildQueryBuilder().forEntity(Trip.class).get()

org.apache.lucene.search.Query lq = qb.keyword().onFields("ticker", "title", "descriptio
org.hibernate.Query ftq = fts.createFullTextQuery(lq, Trip.class);
```

Por último se muestran los datos almacenados en la lista si es que los hubiese.

```
// Recogemos de la query y los mostramos en caso de que se encuentren
List<Trip> ilr = ftg.list();

if (ilr.isEmpty() == false)
    for (Trip t : ilr) {
        System.out.println("\n> Datos encontrados para '" + tecladoString + "'");
        System.out.println("Trip " + t.getTicker() + ", tittle: " + t.getTitle() + ", de
    }
else
    System.out.println("No encontrado");
fts.close();
}
```

Resultados

Intentamos buscar una palabra no indexada

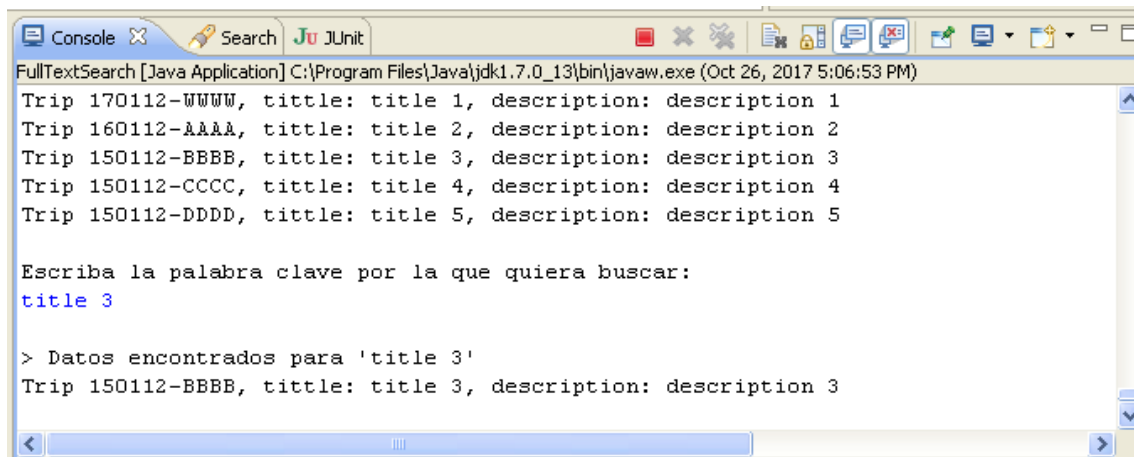
```
Trip 170112-WWWW, tittle: title 1, description: description 1
Trip 160112-AAAA, tittle: title 2, description: description 2
Trip 150112-BBBB, tittle: title 3, description: description 3
Trip 150112-CCCC, tittle: title 4, description: description 4
Trip 150112-DDDD, tittle: title 5, description: description 5
```

Escriba la palabra clave por la que quiera buscar:

p

No encontrado

Buscando una palabra indexada



```
FullTextSearch [Java Application] C:\Program Files\Java\jdk1.7.0_13\bin\javaw.exe (Oct 26, 2017 5:06:53 PM)
Trip 170112-WWWW, tittle: title 1, description: description 1
Trip 160112-AAAA, tittle: title 2, description: description 2
Trip 150112-BBBB, tittle: title 3, description: description 3
Trip 150112-CCCC, tittle: title 4, description: description 4
Trip 150112-DDDD, tittle: title 5, description: description 5

Escriba la palabra clave por la que quiera buscar:
title 3

> Datos encontrados para 'title 3'
Trip 150112-BBBB, tittle: title 3, description: description 3
```

Una vez realizada una búsqueda deberá volver a ejecutar FulltextSearch.java para volver a realizar otra. Tenga en cuenta que al volver a ejecutar mostrara algún warning debido a que ya están indexados algunos datos.

Bibliografía:

<http://hibernate.org/search/documentation/getting-started/>

<https://mprabhat.me/2012/09/30/full-text-search-with-hibernate-search-4-1-lucene-and-jpa/>