Logo 1

Logo2

Universität Bayreuth
Fakultät für Informatik

Bachelorarbeit

im Studiengang Informatik

zur Erlangung des akademischen Grades
Bachelor / Master of Science

Thema:	Integration of JPA-conform ORM-Implementations in Hibernate Search
Autor:	Martin Braun <martinbraun123@aol.com> Matrikel-Nr. 1249080
Version vom:	July 17, 2015
1. Betreuer:	Dr. Bernhard Volz
2. Betreuer:	Prof. Dr. Bernhard Westfechtel

Zusammenfassung

Abstract

Contents

List of Figures

List of Tables

Listingverzeichnis

1 Introduction

1.1 An overview of different database paradigms

When it comes to persisting data in applications, nowadays there exist a lot of different paradigms that one has to choose from. Following is a short explanation for the two currently most used ones.

1.1.1 relational databases

1.1.2 NoSQL databases

1.2 Object-relational impedance mismatch

While the NoSQL approach is undeniably rising in popularity, the demand for relational solutions is still unmatched as it has been used and proven in practice for many years now.

Nowadays, many popular languages like Java, C#, etc. are object-oriented. While SQL solutions for querying relational databases exist for these languages, there is a big discrepancy between the relational and the object-oriented paradigm. [1]

This is where Object Relational Mappers (ORM) come into use. They map tables to entities (classes) and enable users to write queries against classes instead of tables. This is especially useful if used in big software products as not all programmers have to know the exact details of the underlying database. The database system could even be completely replaced for another with the business logic still being in place.

1.2.1 JPA

The first version of the JPA standard was released in May 2006. From then on it rose to probably the most commonly used persistence API for Java. Initially created to standardize only relational database mappers, it since has become the standard for other database concepts like for example NoSQL. The currently newest version of this standard is 2.1.[2]

Some popular implementations are:

- Hibernate ORM
- EclipseLink

[1]Wikipedia on Object relational impedance mismatch, see [1]
[2]Wikipedia on Java Persistence API, see [2]

- OpenJPA

Using the standardized JPA API has some interesting benefits. For one, the specific JPA implementation can be swapped out. This is particularily important if you are working in a Java EE compliant environment. Java EE itself is a specification for platforms, mostly Web-servers.[3] Many Java EE Web-servers ship with a bundled JPA implementation that they are optimized for. This means that if a user switches servers, he/she is also likely to swap out the JPA implementor. If the user's application is strictly JPA compliant, little to no problems will arise upon such a change.

1.3 fulltext search

Conventional relational databases are extremely good at retrieving and querying structured data. But if you want to build a search engine atop your domain model, most RDBMS will only support the SQL-LIKE operator:

```
SELECT book.id FROM book WHERE book.name LIKE %name%;
```

While this might be enough for some applications, this wildcard query doesn't support features a good search engine would need, for example:

- fuzzy queries (variations of the original string will get matched, too)
- phrase queries (search for a specified phrase)
- regular expression queries (matches are determined by a regular expression)

There may exist some RDBMS that support similar query-types, but in the context of using a ORM we would then lose the ability to switch databases since we require specific features not every RDBMS supports.

1.3.1 compatibility of Hibernate Search with JPA

1.4 aims of this thesis

2 Showcase of basic Hibernate Search

3 Ausblick

4 Fazit

Abbildung **??** [S.**??**]

[3]Wikipedia on Java EE, see [3]

Überschrift 1	Überschrift 2
Info 1	Info 2
Info 3	Info 4

```xml
<dataConfig>
  <dataSource type="JdbcDataSource"
              driver="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost/bms_db"
              user="root"
              password=""/>
  <document>
    <entity name="id"
        query="select id, htmlBody, sentDate, sentFrom, subject, textBody
        from mail">
    <field column="id" name="id"/>
    <field column="htmlBody" name="text"/>
    <field column="sentDate" name="sentDate"/>
    <field column="sentFrom" name="sentFrom"/>
    <field column="subject"  name="subject"/>
    <field column="textBody" name="text"/>
    </entity>
  </document>
</dataConfig>
```

Listing 1: Die Datei `data-config.xml` dient als Beispiel für XML Quellcode

```java
/* generate TagCloud */
Cloud cloud = new Cloud();
cloud.setMaxWeight(_maxSizeOfText);
cloud.setMinWeight(_minSizeOfText);
cloud.setTagCase(Case.LOWER);

/* evaluate context and find additional stopwords */
String query = getContextQuery(_context);
List<String> contextStoplist = new ArrayList<String>();
contextStoplist = getStopwordsFromDB(query);

/* append context stoplist */
while(contextStoplist != null && !contextStoplist.isEmpty())
  _stoplist.add(contextStoplist.remove(0));

/* add cloud filters */
if (_stoplist != null) {
  DictionaryFilter df = new DictionaryFilter(_stoplist);
  cloud.addInputFilter(df);
}
/* remove empty tags */
NonNullFilter<Tag> nnf = new NonNullFilter<Tag>();
cloud.addInputFilter(nnf);
```

```

/* set minimum tag length */
MinLengthFilter mlf = new MinLengthFilter(_minTagLength);
cloud.addInputFilter(mlf);

/* add taglist to tagcloud */
cloud.addText(_taglist);

/* set number of shown tags */
cloud.setMaxTagsToDisplay(_tagsToDisplay);
```

Listing 2: Das Listing zeigt Java Quellcode

Die Zuordnung aller möglichen Werte, welche eine Zufallsvariable annehmen kann nennt man *Verteilungsfunktion* von X.

> Die Funktion F: $\mathbb{R} \rightarrow [0,1]$ mit $F(t) = P(X \leq t)$ heißt Verteilungsfunktion von X.[4]
>
> Für eine stetige Zufallsvariable $X : \Omega \rightarrow \mathbb{R}$ heißt eine integrierbare, nichtnegative reelle Funktion $w : \mathbb{R} \rightarrow \mathbb{R}$ mit $F(x) = P(X \leq x) = \int_{-\infty}^{x} w(t)dt$ die *Dichte* oder *Wahrscheinlichkeitsdichte* der Zufallsvariablen X.[5]

[4]Konen, vgl. [?] [S.55]
[5]Konen, vgl. [?] [S.56]

Literaturverzeichnis

Anhang

Literaturverzeichnis

[1] Object-relational impedance mismatch, Wikipedia `https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch`, 07/15/2015

[2] Wikipedia `https://en.wikipedia.org/wiki/Java_Persistence_API`, 07/16/2015

[3] Java Platform, Enterprise Edition Wikipedia `https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition`, 07/16/2015

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur <-Arbeit>

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift:$ $Ort, Datum:$