AURÉLIEN PLISNIER

Advanced Databases Project

Introduction to Java Hibernate

Université libre de Bruxelles
Ecole polytechnique de Bruxelles
Première année du Master en Ingénieur civil en Informatique
Année académique 2013-2014

Prof. Esteban Zimanyi
INFO-H-415

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this project is to give an overview of how and why to use Java Hibernate. A simple application using a MySQL database, extracted from the DBLP project[1], will be used as a tutorial to Java Hibernate and Hibernate Query Language (HQL).
Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database.

1.2 Object-relational mapping

The role of the object-relational mapping (ORM) is to create, from a relational database, a virtual object database that can be accessed in an object oriented code.
The importance of the mapping between the object and the relational paradigms comes from their predominance respectively in the worlds of software engineering and data storage.

Another point is the inner difference between the two paradigms: Scott W. Ambler says "The object paradigm is based on software engineering principles such as coupling, cohesion, and encapsulation, whereas the relational paradigm is based on mathematical principles, particularly those of set theory. [...] the preferred approach to access: with the object paradigm you traverse objects via their relationships, whereas with the relational paradigm you duplicate data to join the rows in tables." [2]

Storing graphs of objects into a relational database exposes us to several mismatch problems[3]:

1. Inheritance in general and subtypes in particular are not managed by relational DBMS;
2. Granularity may be quite different;
3. A RDBMS defines exactly one notion of sameness: the primary key. Java, however, defines both object identity a==b and object equality a.equals(b);
4. In Java, like as in many programming languages, you navigate from one association to another walking the object network, in the same way as using a network CODASYL DBMS such as CA-IDMS. Retrieving data from a RDMS is a set oriented process.

Though these differences make the mapping between object and relational models a tricky thing to do, using ORM gives a lot of advantages. Such as an increased productivity and maintainability by avoiding tedious SQL statements[7], increased maintainability by reducing the number of lines of code and increased portability since the Java code is far less dependent of the SQL database. [11]

The main disadvantage of ORM in general is a (sometimes not affordable) potential performance overhead.

Chapter 2

The choice of Hibernate

2.1 The different solutions

There are a lot of different ORM frameworks: [8] gives a non-exhaustive list of them.

Hibernate emerged in 2002 as a response to lacks in EJB (Enterprise Java Beans) 2 an 2.1 standards. It was an open source solution, in contrary to TopLink which was a proprietary solution from WebGain, bought later by Oracle.
Hibernate inspired the standard JPA (Java Persistence Api) 1.0 which is a part of EJB 3.0 specs. EJB 3.0 is the standard API for Java persistence. In 2010 Hibernate was certified JPA 2.0 complient. [6]
The current Hibernate version is 4.2.7.

TopLink (Fusion, EclypseLink): TopLink for Java emerged in 1998. It was initially a proprietary solution from WebGain. At the time, its main advantages were the relational power and more effectiveness and flexibility than the entity beans but at the cost of a certain complexity. Its main problem was that it was a costly solution, while the Java community was waiting for an open source free unique norm. It was bought by Oracle in 2002 and then incorporated in the Oracle Fusion middleware.
Nowadays, it is integrated in EclypseLink. [6]

Open JPA is an open persistence engine hosted by Apache and used by BEA for its Weblogic software and by IBM for its Websphere. [10]

MyBatis proposes an opposite approach: it creates XML configuration files from SQL statements so that Java objects can be built from a relational structure. [9]

The first question that arise is: how to choose an ORM software amongst them ?

2.2 Why we choose Hibernate

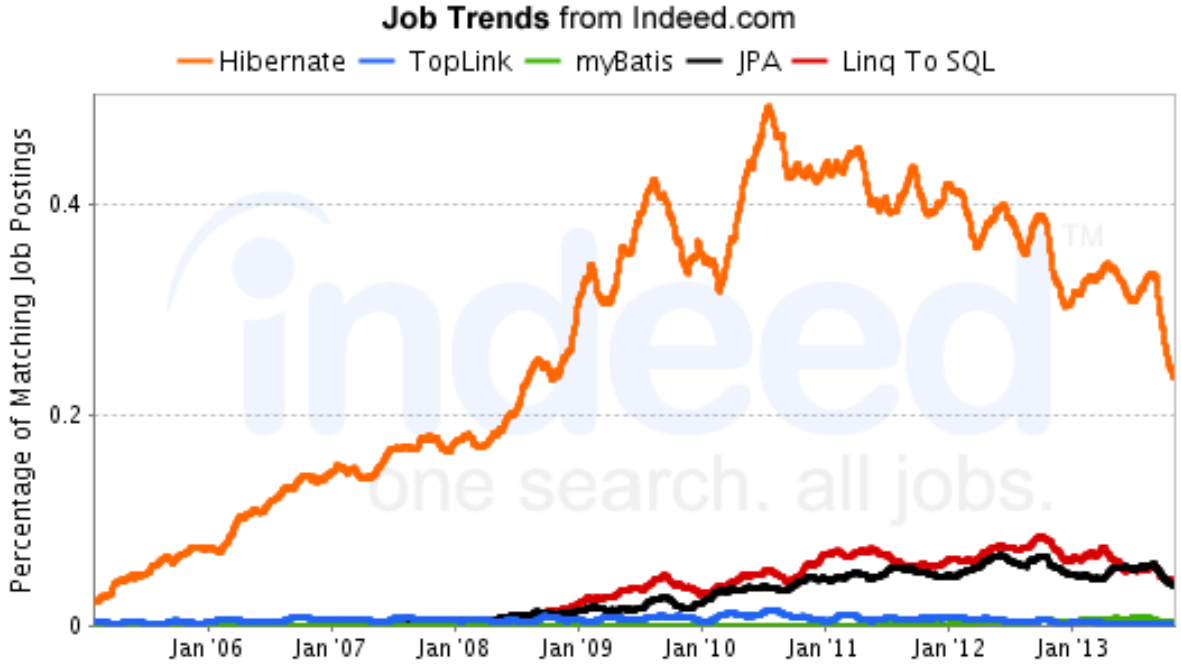
Cost Effective: Hibernate is free and open source.

Short learning curve: Because most programmers have good object oriented experience and Hibernate is a totally object oriented concept, its learning curve is relatively short. [11]

Popular: There are many books, communities and forums about Hibernate. When something goes wrong with Hibernate, you can usually Google and find an answer easily. [11]
Hibernate is the most predominant ORM solution on the market, it has the widest documentation, the most support, the biggest community and the largest available set of experienced developers [12]: below is a graph from Indeed.com showing job vacations trends in the US for some predomiment ORM solutions.[13]

This is an very important criteria for a company : to be sure that it will easily find manpower and support from consultants for its projects; by this way, it becomes also an important factor of the cost of ownership of the solution.

Sustainability: Hibernate is now more than ten years old and, despite of the long list of contenders[8], it remains far more popular than all other frameworks, even those supported by big software companies. This is another major criteria for any client company.



Standard and up-to-date: As mentioned earlier, Hibernate follows the standards very closely, inspiring them if needed. This makes the choice less critical : a later "lift and shift" to another framework would not be too painfull...

2.3 Hibernate

Hibernate solves Object-Relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. Hibernate acts as an SQL adaptor when the application logic handles persisted objects.
The glue between the object world used in Java and the relational world used by the DBMS is implemented with :

- a mapping configuration for each persisted object (called the entity),
- an associated Hibernate implementation to populate/persist each object (called the repository),
- a configuration (datasource, transaction manager) to create a session to handle the objects.

In other words, Hibernate translates relational tables into Java objects. It is a potent alternative to Serialization when it comes to ensuring object persistence.
Hibernate supports the mosts used DBMS's, including DB2, Oracle, MySQL, PostgreSQL and SQL Server.
Another key point is that the Java code of an application using Hibernate may be totally independent from the DBMS used by the application. Allowing to switch from one DBMS to another without changing the code, this is portability (see § 6.1).
Hibernate prepares SQL syntax based on an HQL request and submit it to the DBMS. The "HQL to SQL"

translation can be realized in respect with target DBMS features (Hibernate configuration = dialect). Doing this, Hibernate acts as a performance optimizer.
Hibernate supports native SQL too, but loosing then all the benefits pointed here. It is therefore recommended to use HQL queries as much as possible and only to fall back to SQL queries when HQL doesn't fill the requirements.

Chapter 3

Project's structure

3.1 The different packages

The project is divided into 4 main packages: a "default" package contains the class main and the main mapping XML file (that file has to be in the same package as the main class).
The dblp_objects package contains the Java classes corresponding to the dblp tables we used in this project and all the related XML mapping files.
The queries package contains the hql queries we wrote during the project.
The package userInterface contains the view, built with NetBeans IDE.

3.2 The database

The database used for this project is a part of the DBLP Computer Science Bibliography. The XML file containing the database was parsed and a subset of it was injected into our own relational database, according to INFO-H-303 last year's assignments.

3.2.1 ER Model

Below is the entity relational model of our database. 3.1

3.2.2 The tables

Below is the structure of our relational database.

Table 3.1: Article

Column	Type	Null	Default
Publication_id	int(8)	No	
Volume	varchar(50)	Yes	NULL
Number	varchar(20)	Yes	NULL
Pages	varchar(20)	Yes	NULL
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.2: Author

Column	Type	Null	Default
Author_id	int(8)	No	
DBLP_www_Key	varchar(150)	No	

Table 3.2: Author (end)

Column	Type	Null	Default
URL	varchar(300)	Yes	NULL
Crossref	varchar(150)	Yes	NULL
Note	varchar(150)	Yes	NULL
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.3: Author_Name

Column	Type	Null	Default
Author_id	int(8)	No	
Name	varchar(70)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.4: Author_Publication

Column	Type	Null	Default
Author_id	int(8)	No	
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.5: Book

Column	Type	Null	Default
Publication_id	int(8)	No	
ISBN	varchar(25)	Yes	NULL
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.6: Editor

Column	Type	Null	Default
Editor_id	int(8)	No	
Name	varchar(70)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.7: Editor_Article

Column	Type	Null	Default
Editor_id	int(8)	No	
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.8: Editor_Book

Column	Type	Null	Default
Editor_id	int(8)	No	

Table 3.8: Editor_Book (end)

Column	Type	Null	Default
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.9: Journal

Column	Type	Null	Default
Name	varchar(100)	No	
Year	int(4)	No	0
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.10: Journal_Article

Column	Type	Null	Default
Journal_name	varchar(100)	No	
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.11: PHDThesis

Column	Type	Null	Default
Publication_id	int(8)	No	
ISBN	varchar(25)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.12: Publication

Column	Type	Null	Default
Publication_id	int(8)	No	
DBLP_Key	varchar(150)	No	
Title	varchar(300)	No	
URL	varchar(300)	Yes	NULL
EE	varchar(300)	Yes	NULL
Year	int(4)	No	0
Crossref	varchar(150)	Yes	NULL
Note	varchar(150)	Yes	NULL
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.13: Publisher

Column	Type	Null	Default
Publisher_id	int(8)	No	
Name	varchar(250)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.14: Publisher_Publication

Column	Type	Null	Default
Publisher_id	int(8)	No	
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.15: School

Column	Type	Null	Default
School_id	int(8)	No	
Name	varchar(150)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.16: School_Thesis

Column	Type	Null	Default
School_id	int(8)	No	
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.17: Thesis

Column	Type	Null	Default
Publication_id	int(8)	No	
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.18: User

Column	Type	Null	Default
User_id	int(8)	No	
Email	varchar(100)	No	
Password	char(64)	Yes	NULL
Administrator	int(1)	No	0
Time_stp	timestamp	No	CURRENT_TIMESTAMP

Table 3.19: User_Publication

Column	Type	Null	Default
User_id	int(8)	No	
Publication_id	int(8)	No	
Comment	varchar(300)	Yes	NULL
Time_stp	timestamp	No	CURRENT_TIMESTAMP

3.2.3 Cardinality

Below is the content of the different tables in term of number of rows.

Article 309146

Author	379501
Author_Name	387511
Author_Publication	898582
Book	1619
Editor	266
Editor_Article	1
Editor_Book	316
Journal	4124
Journal_Article	309144
PHDThesis	2884
Publication	313650
Publisher	111
Publisher_Publication	1611
School	169
School_Thesis	2884
Thesis	2885

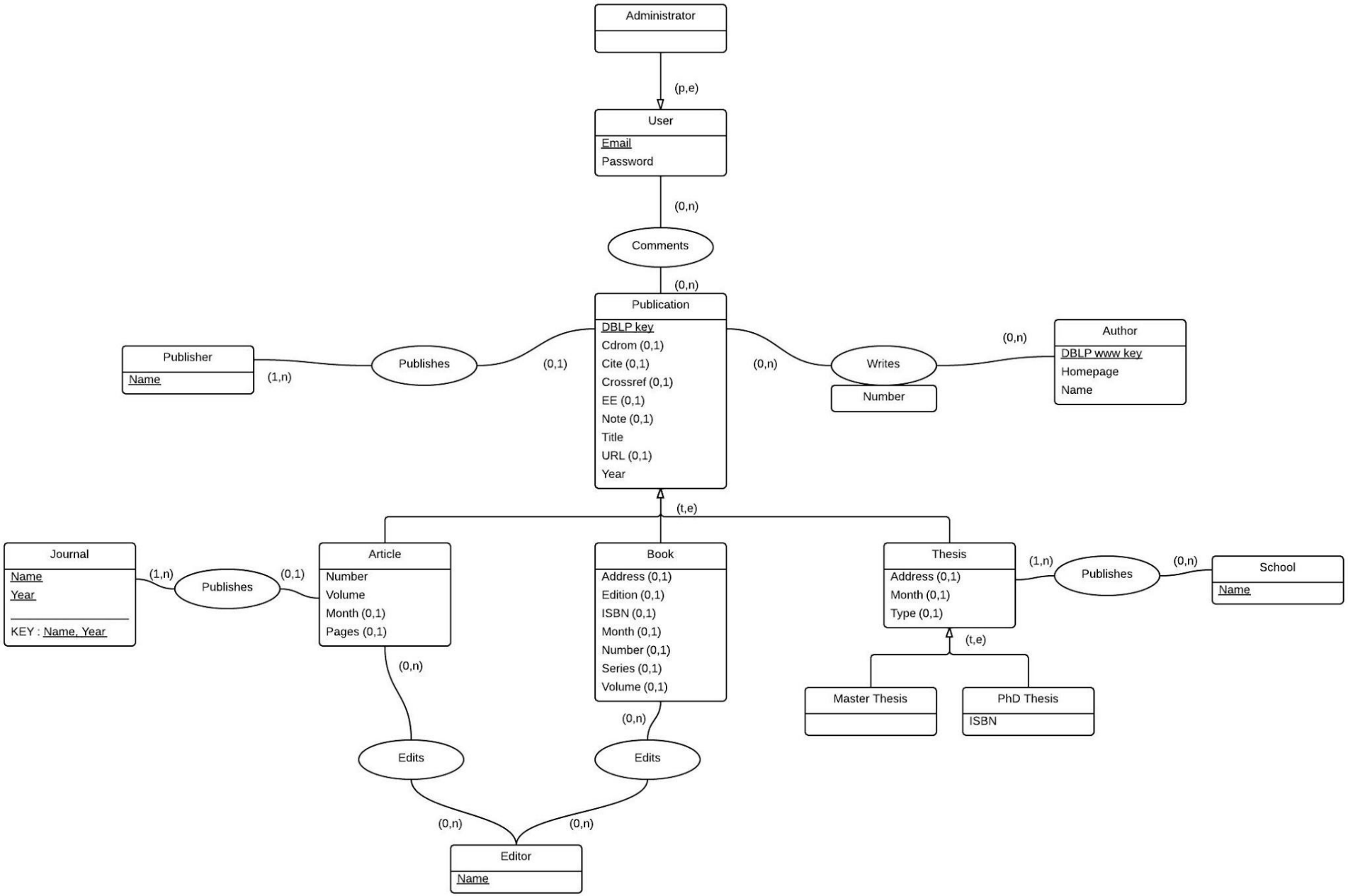


Figure 3.1: ER Model

Chapter 4

Configuring Hibernate and mapping the DB

4.1 Installation

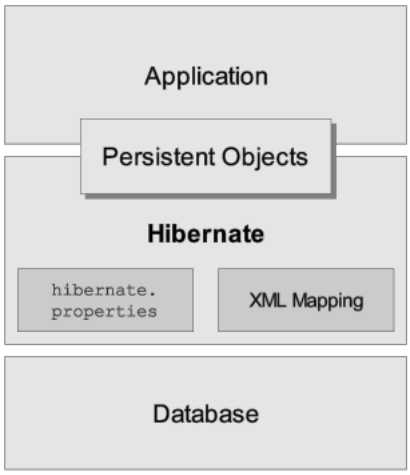
The first step is to download the Hibernate library and to add it to the project's build path. It can be found on www.hibernate.org.

4.2 Mapping declarations

Object/relational mappings can be defined in three approaches:

1. using Java 5 annotations (via the Java Persistence 2 annotations - see Java Spec Req 317);
2. using JPA 2 XML deployment descriptors;
3. using the Hibernate legacy XML files approach known as hbm.xml.

Annotations in the Java code is the most recommandable way to implement mapping because it makes code more independant of Hibernate (better portability) and more readable. But the Hibernate XML descriptors approach remains more powerfull and the only solution in special circumstances: obviously, in most cases, such as the one we tested, the RDB is preexistent to the OO model and not specialy tailored to the needs of the OO model. It is why we choose first to try the third approach and write XML files to configure Hibernate and to make the mapping between Java classes and SQL tables. Doing this way, the mapping declarations and the Java code are separated.
Below is an sketch of the software architecture.[4]



4.3 The file hibernate.cfg.xml

This is the configuration file.
First we describe the Hibernate version used and the DBMS. Then the connection properties.
In the example file we use MySQL, the database is called dbms and is in localhost, the username is root and there is no password.
Setting the property show_sql to true makes your application print the automatically generated SQL queries in the terminal during the execution.
Lastly we need to indicate all the mapping files.
See [14] for another example.

4.4 Mapping a class using a XML descriptor

hbm.xml files do the mapping between the Java classes and the tables.
The mapping is indicated by the tag <hibernate-mapping>.
The name of the class is mapped to the name of the table inside the tag <class>.
The key is mapped in the tag <id>.
All the other atributes are mapped in separate tags <property>.
Many to many relationship is illustrated into publication.hbm.xml with the <set> tag.
Inheritance is mapped using the tag <joined-subclass> inside which the subclass is mapped to the corresponding table. Multiple <joined-subclass> can be nested to map multi-level inheritance.

4.5 Writing the Java Classes

To be mapped, Java classes must at least contain getter and setter for each attribute and an empty constructor. This is the only restriction.

Chapter 5

Querying

Queries can be found in the package "queries" of the project.

5.1 Basic queries

Some basic queries were made to illustrate basic functions: the class SigninQuery illustrates the insertion, DeleteUserQuery the deletion.
Basic searches are illutrated by BookSearchQuery, LoginQUery and UserSearchQuery.
The use of the result of a query as input for a next query is illustrated in BookDetailQuery. These queries use the main methods found in figure 5.1.

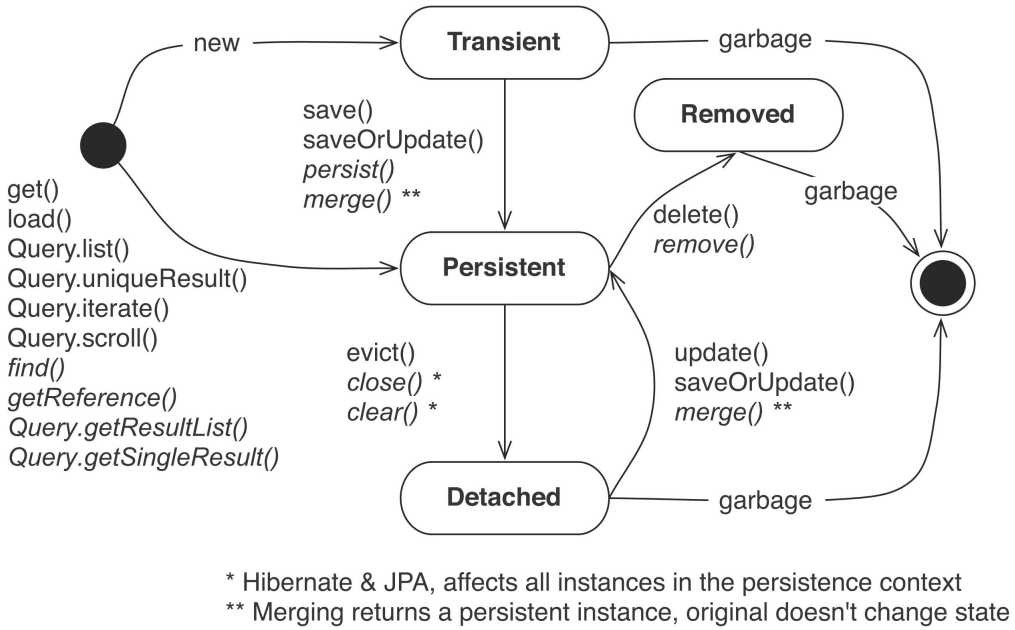


Figure 5.1: Object states and their transitions[5]

5.2 The special requests

To illustrate the use of Hibernate and HQL, we implemented 6 special requests. They are taken from the assignements of 2012-2013's INFO-H-303 - Databases project.
The special requests were implemented in the package queries of the source code. For each request we made a personal SQL query and compared its results with the one generated by Hibernate.
Simple performance comparisons were made to compare the time taken by beginner-written SQL and SQL generated by Hibernate from beginner-written HQL.

5.2.1 Request 1

Retrieve all the authors who published at least once every year between 2008 and 2010 included.

Hand made SQL request :

```
SELECT Author_id
FROM Publication p , Author_Publication a , Author
WHERE p . Publication_id = a . Publication_id AND p.Year >= 2008 AND p.Year <= 2010
AND a.Author_id = Author.Author_id
GROUP BY a.Author_id
HAVING COUNT(distinct p.Year ) = 3
```

Hand made HQL request :

```
select A.author_id
from publication as P
join P.authors as A
where P.year <= 2010 and P.year >= 2008
group by A.author_id
having count(distinct P.year) = 3
```

SQL generated by Hibernate :

```
select author2_.Author_id as col_0_0_
from publication publicatio0_
inner join author_publication authors1_ on publicatio0_.Publication_ID=authors1_.publication_id
inner join author author2_ on authors1_.author_id=author2_.Author_id
where publicatio0_.Year<=2010
and publicatio0_.Year>=2008
group by author2_.Author_id
having count(distinct publicatio0_.Year)=3
```

Performance comparison :
Hand made: 2.39 sec.
Hibernate generated: 2.41 sec.

5.2.2 Request 2

The authors who wrote at least two articles during the same year.

Hand made SQL request :

```
SELECT DISTINCT AP1.Author_id
FROM Author_Publication AP1 , Author_Publication AP2 , Article AR1, Publication P1 , Publica-
tion P2 , Article AR2
```

WHERE AP1.Author_id = AP2.Author_id
AND AR1.Publication_id = P1.Publication_id
AND P1.Publication_id = AP1.Publication_id
AND AR2.Publication_id = P2.Publication_id
AND P2.Publication_id = AP2.Publication_id
AND P1.Year = P2.Year
AND P1.Publication_id != P2.Publication_id
ORDER BY AP1.Author_id

Hand made HQL request :

Select Distinct Au.author_id
From article as Ar
Join Ar.authors as Au
Group by Au.author_id, Ar.year
Having count(Distinct Ar.publication_ID) >= 2

SQL generated by Hibernate :

select distinct author2_.Author_id as col_0_0_
from article article0_
inner join publication article0_1_ on article0_.publication_ID=article0_1_.Publication_ID
inner join author_publication authors1_ on article0_.publication_ID=authors1_.publication_id
inner join author author2_ on authors1_.author_id=author2_.Author_id
group by author2_.Author_id , article0_1_.Year
having count(distinct article0_.publication_ID)>=2

Performance comparison :

Hand made: 31.53 sec.
Hibernate generated: 17.98 sec.

5.2.3 Request 3

The authors Y at a distance 2 of a given author X.
An author Y is at a distance 1 of an author X if they wrote an article together.

Hand made SQL request :

SELECT distinct ap4.Author_id
FROM Author_Publication ap1 , Author_Publication ap2 , Author_Publication ap3 , Author_Publication ap4 , Article ar , Article ar2
WHERE ap1.Author_id = 1071543
AND ap1.Publication_id = ap2.Publication_id
AND ap3.Publication_id = ap4.Publication_id
AND ap1.Publication_id != ap3.Publication_id
AND ap1.Author_id != ap2.Author_id
AND ap3.Author_id != ap4.Author_id
AND ap1.Author_id != ap4.Author_id
AND ap2.Author_id = ap3.Author_id
AND ar.Publication_id = ap1.Publication_id
AND ar2.Publication_id = ap3.Publication_id
ORDER BY ap4.Author_id

Hand made HQL request :

Select distinct A3.author_id
From article as P1, article as P2, article as P3
Join P1.authors as A1
Join P2.authors as A2
Join P3.authors as A3

```
Where A1.author_id = 1071543
And P1.publication_ID != P2.publication_ID
And P1.publication_ID != P3.publication_ID
And P2.publication_ID != P3.publication_ID
And A1.author_id != A2.author_id
And A1.author_id != A3.author_id
And A2.author_id != A3.author_id
And A2.author_id in (Select author_id From P1.authors)
And A2.author_id in (Select author_id From P3.authors)
Order by A3.author_id
```

SQL generated by Hibernate :

```
select distinct author8_.Author_id as col_0_0_
from article article0_ inner join publication article0_1_ on article0_.publication_ID=article0_1_.Publication_ID
inner join author_publication authors3_ on article0_.publication_ID=authors3_.publication_id
inner join author author4_ on authors3_.author_id=author4_.Author_id, article article1_
inner join publication article1_1_ on article1_.publication_ID=article1_1_.Publication_ID
inner join author_publication authors5_ on article1_.publication_ID=authors5_.publication_id
inner join author author6_ on authors5_.author_id=author6_.Author_id, article article2_
inner join publication article2_1_ on article2_.publication_ID=article2_1_.Publication_ID
inner join author_publication authors7_ on article2_.publication_ID=authors7_.publication_id
inner join author author8_ on authors7_.author_id=author8_.Author_id
where author4_.Author_id=1071543
and article0_.publication_ID<>article1_.publication_ID
and article0_.publication_ID<>article2_.publication_ID
and article1_.publication_ID<>article2_.publication_ID
and author4_.Author_id<>author6_.Author_id
and author4_.Author_id<>author8_.Author_id
and author6_.Author_id<>author8_.Author_id
and (author6_.Author_id in (
select author10_.Author_id
from author_publication authors9_, author author10_
where article0_.publication_ID=authors9_.publication_id
and authors9_.author_id=author10_.Author_id))
and (author6_.Author_id in (
select author12_.Author_id
from author_publication authors11_, author author12_
where article2_.publication_ID=authors11_.publication_id
and authors11_.author_id=author12_.Author_id))
order by author8_.Author_id
```

Performance comparison :

Hand made: 0.0022 sec.
Hibernate generated: 0.0042 sec. without order by

5.2.4 Request 4

Articles without any doctor amongst their authors. A doctor is defined as an author who wrote at least one phd thesis alone.

Hand made SQL request :

```
SELECT ar2.Publication_id
FROM Article ar2 , Publication p
WHERE ar2.Publication_id = p.Publication_id
AND ar2.Publication_id NOT IN (
```

```
SELECT ap3.Publication_id
FROM Author_Publication ap , PHDThesis pt , Author_Publication ap2 ,
Author_Publication ap3 , Article ar
WHERE ap.Publication_id = pt.Publication_id
AND ap2.Publication_id = ap.Publication_id
AND ap3.Publication_id = ar.Publication_id
AND ap3.Author_id = ap2.Author_id
GROUP BY ap.Publication_id
HAVING COUNT(*)=1 )
Order By ar2.Publication_id
```

Hand made HQL request :

```
Select Distinct Ar.publication_ID
From article as Ar
Join Ar.authors as Au
Where Au.author_id Not In (
        Select A.author_id
        From phdThesis T
        Join T.authors as A
        Group By T.publication_ID
        Having Count(Distinct A.author_id) = 1)
Order By Ar.publication_ID
```

SQL generated by Hibernate :

```
SELECT DISTINCT article0_.publication_ID AS col_0_0_
FROM article article0_
INNER JOIN publication article0_1_ ON
        article0_.publication_ID = article0_1_.Publication_ID
INNER JOIN author_publication authors1_ ON
        article0_.publication_ID = authors1_.publication_id
INNER JOIN author author2_ ON
        authors1_.author_id = author2_.Author_id
WHERE author2_.Author_id NOT
IN (
SELECT author5_.Author_id
FROM phdThesis phdthesis3_
INNER JOIN thesis phdthesis3_1_ ON
        phdthesis3_.publication_ID = phdthesis3_1_.publication_ID
INNER JOIN publication phdthesis3_2_
        ON phdthesis3_.publication_ID = phdthesis3_2_.Publication_ID
INNER JOIN author_publication authors4_
        ON phdthesis3_.publication_ID = authors4_.publication_id
INNER JOIN author author5_
        ON authors4_.author_id = author5_.Author_id
GROUP BY phdthesis3_.publication_ID
HAVING count( DISTINCT author5_.Author_id ) =1
)
ORDER BY article0_.publication_ID
```

Performance comparison :

Hand made: 0.64 sec.

Hibernate generated: 7.02 sec.

5.2.5 Request 5

The author who published in the greatest number of different journals.

Hand made SQL request :

```
SELECT an .Name, COUNT(Distinct ja.Journal_Name ) cpteur
FROM Author_Publication ap , Author a , Journal_Article ja , Author_Name an
WHERE ja.Publication_id=ap.Publication_id
AND ap.Author_id=a.Author_id
AND a . Author_id=an . Author_id
GROUP BY a.Author_id
ORDER BY cpteur DESC
LIMIT 1
```

Hand made HQL request :

The command "query.setMaxResults(1);" was used before executing the query, this is how limit(1) is achieved with HQL.

```
Select Distinct A.author_id
From journal as J
Join J.articles as P
Join P.authors as A
Group By A.author_id
Order By Count(Distinct J.name) desc
```

SQL generated by Hibernate :

```
select distinct author4_.Author_id as col_0_0_
from journal journal0_
inner join journal_article articles1_ on
        journal0_.Name=articles1_.journal_name
inner join article article2_ on
        articles1_.Publication_id=article2_.publication_ID
inner join publication article2_1_ on
        article2_.publication_ID=article2_1_.Publication_ID
inner join author_publication authors3_ on
        article2_.publication_ID=authors3_.publication_id
inner join author author4_
        on authors3_.author_id=author4_.Author_id
group by author4_.Author_id
order by count(distinct journal0_.Name) desc
limit 1
```

Performance comparison :

Hand made: 85.40 sec.
Hibernate generated: 84.69 sec.
Returned ID: 1053195, with 80 journals.

5.2.6 Request 6

The journals with their total number of articles, the average number of articles per year and the average number of authors per article since the journal's year of creation, for journals having a greater number of volumes per year than the average number of volumes per year for the journals.

Note that Hibernate does not take away the possibility to directly use SQL queries into the code. That's what we did here.

Hand made SQL request :

```
SELECT KeptJournal ,
COUNT(Distinct ja3.Publication_id ) AS NumberArticle ,
COUNT( Distinct ja3.Publication_id )/COUNT( Distinct j3.Year )
AS AvgArticlePerYear ,
(
SELECT COUNT( ap.Author_id )/COUNT( Distinct ap.Publication_id )
FROM Journal_Article ja4 , Author_Publication ap
WHERE ja4.Journal_name = KeptJournal
AND ja4.Publication_id = ap.Publication_id
) AS AvgAuthorPerArticle
FROM Journal_Article ja3 , Journal j3 ,
(
SELECT ja2.Journal_name AS KeptJournal
FROM Article ar2 , Journal_Article ja2 , Journal j2 ,
(
SELECT AVG( VolCount ) AS Average
FROM
(
SELECT COUNT( Distinct ar.Volume)/COUNT( Distinct j.Year )*4 AS VolCount
FROM Article ar , Journal_Article ja , Journal j
WHERE ar.Publication_id = ja.Publication_id
AND j.Name = ja.Journal_name
GROUP BY ja.Journal_name
) Volume_Amount_For_Each_Journal
) av
WHERE ja2.Publication_id = ar2.Publication_id
AND j2.Name = ja2.Journal_name
GROUP BY ja2.Journal_name , Average
HAVING COUNT(distinct ar2.Volume)/COUNT( Distinct j2.Year )*4 > Average
) Journals
WHERE ja3.Journal_name = KeptJournal AND j3.Name = KeptJournal
GROUP BY KeptJournal
```

The execution time of that query is 8.98 sec.

5.2.7 Discussion and findings

HQL code seems to be shorter and clearer in general compared to the related SQL code. The fact that HQL is an OQL makes it a higher level language. It felt in general easier to write than SQL, except for request 6: we didn't find how to embed intermediary variables into HQL query.
The translation made by Hibernate from HQL to SQL is very literal. Bad HQL will be translated into bad SQL. That explains why most of the performance results are so close: we first wrote SQL queries, then translated them (except for request 2) into HQL code which was then translated by Hibernate into SQL very similar to the hand made SQL query.

5.2.8 Criteria Queries

We already illutrated two ways to query a database using Hibernate: through HQL or SQL. The main downside of those two solutions is that the execution time of the query is very dependent of the pro-

gramer's skill and comprehesion of SQL mechanics.
Hibernate offers a third way to query databases called criteria queries. It is totally abstracted from SQL. Compared to HQL, it seems to be "less relational and more object oriented".
For this project, HQL was chosen over criteria queries regarding to our previous SQL experience. A future step for this project would be to remake the above queries using criteria queries and compare the performance with the hand written HQL. Criteria queries are well documented by [15].

Chapter 6

Always Hibernate ?

6.1 Hibernate is not only an ORM

As a matter of fact, Hibernate is a piece of software that *enforces independance of the application programs from the DBMS*[1]. Despite of the ANSI standards, each DBMS comes with its idiosyncrasies. Using a framework such as Hibernate brings more portability between several DBMS. This may become a major source of costs reduction when a conversion from one DBMS to another is needed.
As examples of discrepancies between RDBMS, we list here some important differences between the two major contenders on the RDMS market : DB2 and Oracle.[16]

- Clauses for retrieval efficiency such as "OPTIMIZE for nnn ROWS" and "For Fetch Only" are unknown in Oracle.
- The DB2 set statement "EXEC SQL SET :HOST_VARIABLE = ..." must be changed in "EXEC SQL SELECT ... INTO :HOST_VARIABLE FROM DUAL".
- For DB2, an empty string variable is padded with blanks when compared to another; in Oracle, an empty string is NULL and any comparison with a string crashes: ”, ’ ’ have same value for DB2, are different for Oracle; this makes conversion particularly difficult when programs use SUBSTR() function on VARCHAR fields!
- Date, time, and their variants differ more than any other data types between DB2 and Oracle.[17] In particular, in Oracle data type "date" includes "date" and "time" DB2 data types. But the biggest problem comes from the internal format of the timestamps : DB2 is smart enough to recognize the format used in a field and to makes implicit conversions, where Oracle requires explicit declaration.
- About the management of read locks, very important to avoid for example concurrent updates, Oracle supports ANSI/ISO SQL92 standards Read committed and Read serializable, but not Read uncommitted and Repeatable read. In DB2 you have :

[1] If plain native SQL is not used.

1. Repeatable Read (RR) (= SERIALIZABLE in ANSI) (hibernate.connection.isolation=8)
2. Read stability (RS) (= Repeatable Read in ANSI) (hibernate.connection.isolation=4)
3. Cursor stability (CS) (= Read commited in ANSI) (hibernate.connection.isolation=2)
4. Uncommited Read (= Read uncommited in ANSI) (hibernate.connection.isolation=1)

- HQL is great when using SQL functions, whose names and parameters differ very often from one RDBMS to another; examples with DB2 and Oracle:
  1. UCASE() becomes UPPER() for Oracle;
  2. TRANSLATE(EXPR, TO_STRING, FROM_STRING) becomes TRANSLATE(EXPR, FROM_STRING, TO_STRING) !
  3. "date() + 1 month" becomes "add_months(to_date(), 1)";
  4. "cast(7.6 as integer)" does rounding off in Oracle, where it does not in DB2.

6.2 When not use an ORM framework

With our querying tests, we demonstrated that in such circonstances ("complex" queries on one to one mapped OO model/RDB schema), an ORM does not bring much support and gains.

It is surely also the case when we have to initial load a RDB from sequential fixed length input files: neither input nor output are OO.

Bibliography

[1] http://www.informatik.uni-trier.de/ ley/db/

[2] Scott W. Ambler *Mapping objects to relational databases What you need to know and why.* 2000: IBM.

[3] http://hibernate.org/orm/what-is-an-orm/

[4] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole, Hardy Ferentschik *Hibernate Reference Documentation.*

[5] Christian Bauer, Gavin King *Java Persistence with Hibernate* 2007: Manning

[6] Scott W. Ambler *Java Persistence et Hibernate.* 2008: Eyrolles.

[7] http://fr.wikipedia.org/wiki/Mapping_objet-relationnel

[8] http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software#Java

[9] http://en.wikipedia.org/wiki/MyBatis

[10] http://openjpa.apache.org/

[11] http://www.mkyong.com/hibernate/why-i-choose-hibernate-for-my-project/

[12] http://java.dzone.com/news/hibernate-best-choice

[13] http://be.indeed.com/?r=us

[14] http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/ch01.html#d5e83

[15] http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html

[16] *Oracle Database Conversion Guide for DB2 to Oracle9i*. 2004: Oracle

[17] *Dates in DB2 and Oracle: Same Data Type, Different Behavior* http://www.devx.com/dbzone/Article/28713