Université libre de Bruxelles

Advanced Databases

Winter Semester 2017-2018

Object-relational mapping tools and Hibernate

Authors:
Bruno Baldez Correa
Matricule ULB: 000456738
Yue Wang
Matricule ULB: 00000000

Supervisor:
Prof. Esteban Zimanyi

December 20, 2017

Contents

1 Object-relational database

The idea of object-relational database was raised in early 1990s. A research team add object-oriented concepts to relational database. Early commercial products appeared in mid-1990s. IBM, Oracle and some other companies developed early products like Illustra and UniSQL and so on. [8]

The object-relational database (ORD) is a combination of object database and relational database (RDB). It inherits advantages from both database models like a middle man. It supports the basic components of any object-oriented database model in its schemas and the query language used, such as objects, classes and inheritance. It is like a relational database but with an object-oriented database model. In ORD, the basic approach is based on RDB, since the data is stored in a traditional database and manipulated and accessed using queries written in a query language like SQL. [12]

The aim of this technology is to act as a bridge to link conceptual data modeling techniques for relational and object-oriented databases like the entity-relationship diagram (ERD) and object-relational mapping (ORM).

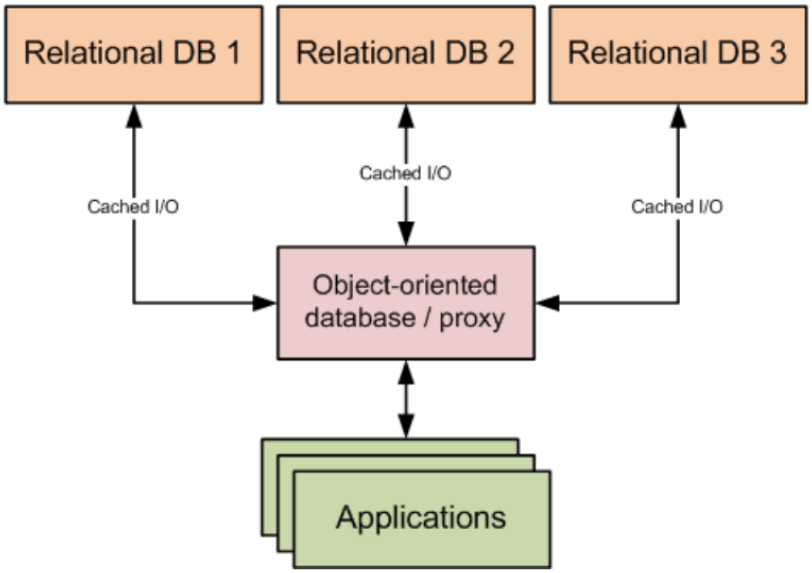


Figure 1: Object-oriented database. Source: [7]

1.1 Necessity of object relational database

When it comes to the necessity of ORG, first it is necessary to clarify the disadvantages of mere relational database and object-oriented database.

The first drawback of relational database is because of its dominant relational language SQL-92. It has limited support to different types of variables, mostly

only numbers and strings while the needs of object type were becoming more important. Another problem is the structure. Relational database provides flat table structure and few supports to complicated structures like nested structures in order to do better query. Also, the trend of software engineering is to use object oriented methods, which brings a need to adopt object oriented ideas into database.

Using object-oriented database can be an option. However, other concerns may raise as well. Object-oriented database comes along with object-oriented languages. Though it can integrate better with OO languages, it may be a problem in return to tie database too closely with application. Also, there is not a certain standard of design model for this way of database development.

1.2 Evaluation of object relational database

Object relational database which tries to integrate advantages of above two databases but exclude the disadvantages can be a good choice. It adopts SQL language but adds many new variable types by using general object oriented languages like C++ or Java, which means it allows users to introduce new class as types. ORD also has object-oriented features like inheritance and polymorphism. In all, ORDs extend the functionality of DBMS products significantly, and an information system using an ORDBMS can be deployed over multiple machines.[3]

To evaluate, some criteria are considered. We evaluate the advantages and disadvantages in terms below.

1.2.1 Extensibility

Object Relational Database has a good extensibility to adding new variable types by using features of object-oriented methods. For example, complex data types with characteristic data types, data specification by texting, new data type creation and user-defined types are all allowed and possible. [11]

1.2.2 Reusability

Another advantage of object relational database is the reusability. It is very easy for users to reuse their codes to some other applications, such as data types, objects, class libraries, even without data access code. This is because instead of coding in each application, users can define the types in the database itself.[13]

1.2.3 Productivity

In the development of application using object relational database, you could save the time for data access code. Although the whole amount of the code may go up, but the data access codes are automatically generated according to the model your defined. [1]

1.2.4 Performance

The performance of object relational database may be the main issue because of this kind of database. Because it generates data access code directly and those codes are sometimes even more complicated than your hand-write codes. Also it may take more time for the data interaction between application layer and database. Though it is a small portion of the whole, a test whether it is acceptable for your application is still necessary.

1.3 Comparison with object database and relational database

To sum up the advantages and disadvantages between object database, relational database and object-relational database, here is a table of the analysis in different dimensions.

	Object Database	**Relational Database**	**Object-Relational Database**
Extensibility	Complex Data types; User self-defined types	Simple data types	Complex Data types; User self-defined types
Reusability	Data types can be reused	None	Data types can be reused
Query Languages	Good integration with programming languages	SQL, a powerful query language	Good integration with programming languages and SQL
User Experience	No standard modeling style, hard to measure	Well developed model system, easy to follow	Well developed model
Performance	Data access codes, taking more time	Direct query which is more efficient	Using mapping tools
Security	No security support	Strong security support	Strong security support based on relational database

Table 1: Databases comparison table

2 Object-relational mapping

In between the concepts of object-relational database, which were explained in the previous section, we can visualize the need for technologies, techniques and tools to support the interpretation of objects created within the program code and their relationship with the relational database. Object-relational mapping, also known as its acronym ORM, is the name given to the layer that uses these tools, technologies and techniques to map data objects in the programming language/application to relations and tables of the database.

Accordingly to King and Bauer [3], ORM, works by transforming data from one representation to another what implies in some performance lose. However, with ORM implemented as middleware, developers would have many opportunities to optimize it what therefore hand-coded SQL layers would not offer. With management and provision of the metadata responsible for the transformation, there gain in development time, but it became uncountable once there is less costs involved than maintaining a hand-coded solution. In this way, an ORM solution consist of four pieces, figure 2.

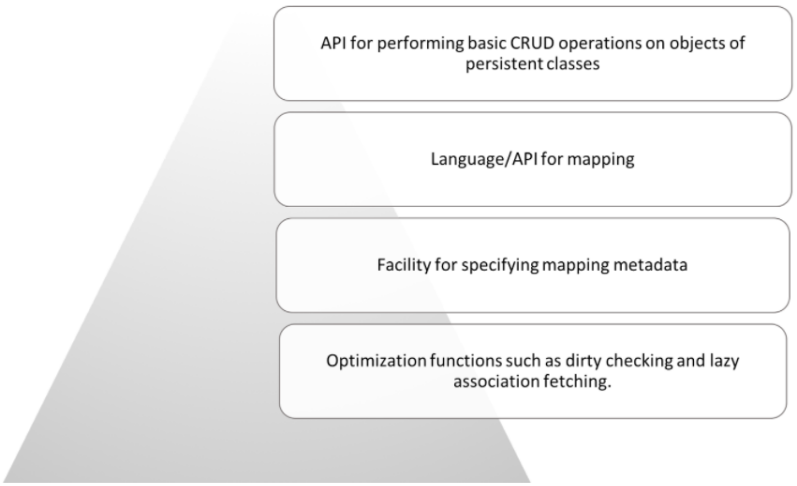


Figure 2: ORM solution pieces. Source: Based on [3]

2.1 ORM Paradigm Mismatch Problems

In front of talking about examples of mapping tools and entity mapping in the application, is necessary to understand some of the known problems about the ORM that were common before the development of tools that nowadays are capable to handle it.

Problem	Description
Granularity	Granularity in relational databases is limited and can only be implemented at two levels.
Subtypes	As inheritance is widely used in object oriented languages is necessary to understand that generally relational databases do not completely support it and when they do they do not adopt a standard syntax.
Identity	Even that in the database tables each tuples represent a unique record in application level is possible that two of those are considered equal by comparing methods of the programming language.
Associations	In application, associations are represented by objects references while in relational databases there is need to do it by using foreign keys.
Data navigation	In SQL this requires joins and selects while in the application it can be done by gettes and setters methods.

Table 2: Paradigm mismatch problems. Source: Based on [15]

2.2 Entity Mapping

Entity, which is identified by an identity, is a persistent domain object. The relationship between entity and table is that an entity class maps to a table in database. Correspondingly, each instance of the entity is a tuple of the table.

From Thomas Stoltmann [10], an Entity refers to a logical collection of data that can be stored or retrieved as a whole, and it follows the these requirements:

- Annotated with the javax.persistence.Entity annotation
- Have a public or protected, no-argument constructor
- Not be declared final, including methods or persistence instance variables
- May extends both entity and non-entity classes
- Persistence instance variables (fields) must be declared private, protected or package-private, and can be accessed directly by the entity class methods

There are four different kinds of mappings: @OneToOne; @ManyToOne; @OneToMany; @ManyToMany. The @JoinyColumn and @JoinTable mappings can be used to join columns and tables

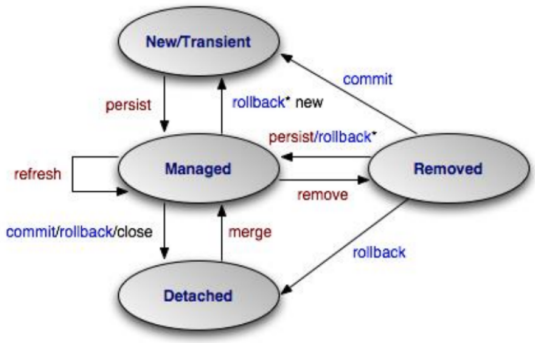


Figure 3: Entity's lifecycle. Source: [10]

2.3 Java Persistence API (JPA)

As object-oriented data models are widely used, the mapping of object-oriented entities to relational database tables (ORs) has been increasingly seen in the design and development of various types of applications [6]. The JPA (Java Persistence API) is a standard developed by Sun Microssystems for mapping Java data objects to relational database objects. JPA makes up for JDBC, ORM, EJB2 and other Java object persistence deficiencies, and is very easy to use. JPA requires extensive support for new features in Java languages such as annotations and generics, and requires J2SE 1.5 (also known as Java 5) or later support.

JPA defines a set of classes and interfaces for implementing persistence management and object / relational mapping. The diagram below shows the main components of JPA and their interrelationships [2].

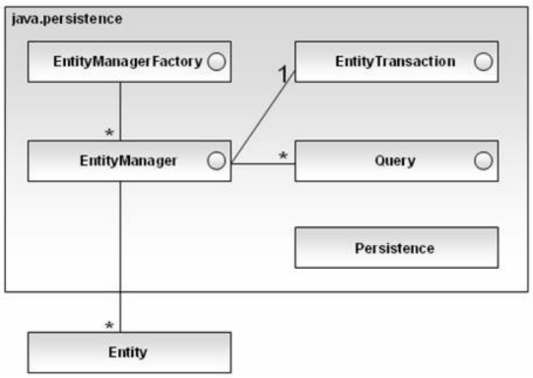


Figure 4: Main components and their relationship. Source: [6]

The figure above can be interpreted as the following:

- **EntityManagerFactory** is the factory class of EntityManager, responsible for creating EntityManager object.
- **EntityManager** is the basic object used in JPA applications, which provides the corresponding method that can manage persistent objects. You can also create or delete persistent objects. EntityManager is also responsible for creating Query instances. When it is used outside of a container, there is a one-to-one relationship between the EntityManagerFactory and the EntityManager.
- **EntityTransaction** Entity operation provides the necessary transaction management, and EntityManager is a one-on-one relationship. You do not need to use EntityTransaction in the query operation, but in the case of object persistence, status updates, object deletion, etc., you must manage the transaction using explicit methods that use EntityTransaction.
- **Query** is the interface of the query entity. Query object can be obtained from the EntityManager. According to the EJB 3.0 specification, Query interface needs to support both JPQL and native SQL syntax.
- **Persistence** is a utility class that creates EntityManagerFactory objects based on parameters provided by the configuration file.

2.3.1 JPA advantages

JPA standard development process has fully absorbed all the advantages of all the persistence technologies that have emerged so far, leaving aside their limitations and making JPA stand out for its ease of use and query capabilities [2].

- **Standardization:** JPA is one of the Java EE standards promulgated by the JCP organization. So any framework that claims to conform to the JPA standard follows the same architecture and provides the same access API, which ensures enterprise applications based on JPA development can be changed with minor changes JPA framework to run.
- **Support for container-level features:** JPA framework support for big data sets, transactions, concurrency and other container-level transactions, which makes JPA beyond the limitations of simple persistence framework, play a greater role in enterprise applications.
- **Easy to use, easy to integrate:** One of JPA's main goals is to provide a simpler programming model: creating entities in the JPA framework is as easy as creating Java classes without any restrictions or restrictions, requiring only annotations using javax.persistence.Entity; JPA's framework and interfaces are also very simple, without too many special rules and design patterns, developers can easily grasp. JPA is based on non-intrusive design and can be easily integrated with other frameworks or containers.

- **Great capabilities of query:** JPA defines a unique JPQL (Java Persistence Query Language), JPQL is an extension of EJB QL, it is a query language for the entity, the operation object is an entity, rather than a relational database table, and can support batch updates and Modify, JOIN, GROUP BY, HAVING usually only SQL can provide advanced query features, and even can support subquery.
- **Advanced features of object-oriented support:** JPA can support advanced object-oriented features, such as inheritance between classes, polymorphism and complex relationships between classes, this support allows developers to maximize the use of object-oriented model of enterprise applications design, and do not need their own The persistence of these features in the relational database is handled.

3 Object-relational mapping tools

The transparency concept in ORM tools, according to Bodden [9], is still very vague. Most tools are not truly transparent but provide a separation of objects persistency behind very simple object-oriented constructs. Object-relational Mapping tools remove mapping classes to databases complexity. This tools provide interfaces which can automatically do selects, updates, inserts and deletes on tables in the database system reflecting changes made to the object model. Below are some examples of the mapping tools using Java. We select three examples out of a long list. Especially we will adopt one of them, Hibernate, to implement our use case.

3.1 Open JPA

Apache's OpenJPA is one of the most popular implementations of Java persistence today. Apache OpenJPA is a Java persistence technology under the Apache Software Foundation that can be used as a standalone POJO (Plain Old Java Object) or integrated into any Java EE-compliant container or other lightweight framework (Such as Tomcat, Spring) and so on together [2].

3.1.1 Data remote transmission / offline processing

The JPA standard provides operating environments that are "local" and "online." Local means that the EntityManager in the JPA application must be directly connected to the specified database, and must be in the same JVM as the code that uses it. Online means that all entity-specific operations must be run within an EntityManager scope. These two features, combined with the fact that EntityManager is nonserialized and can not be transported over the network, which causes JPA applications fail to adapt to the C / S implementation pattern in enterprise applications. OpenJPA extends this part of the interface to support remote data transfer and offline processing.

3.1.2 Database / object view unified tool

When using OpenJPA to develop enterprise applications, keeping database and object view consistency is a very important task. OpenJPA supports three modes to handle database and object view consistency: forward mapping, reverse mapping, Meet-in-the-Middle Mapping, and provides them with the appropriate tools to support them.

- Forward mapping refers to using the org.apache.openjpa.jdbc.meta.MappingTool tool provided in the OpenJPA framework to generate the corresponding database tables from the developer-provided entities and the object / relational mapping annotations provided in the entity.
- Reverse mapping refers to the org.apache.openjpa.jdbc.meta.ReverseMappingTool tool provided in the OpenJPA framework that generates JPA-compliant

entities and corresponding object / relational mapping annotations from database tables.

- The middle match is the developer responsible for creating the database table, JPA-compliant entities, and the corresponding object / relational mapping annotations, using the org.apache.openjpa.jdbc.meta.MappingTool tool provided in the OpenJPA framework to verify that both are consistent.

3.1.3 Use cache to improve efficiency

Performance is one of the focuses of enterprise applications, and caching is one of the important means to improve the performance of enterprise systems. OpenJPA provides a variety of levels and aspects of the cache support for data persistence, including data, queries, assembler query cache. These caching applications can greatly improve the operational efficiency of enterprise applications.

3.2 EclipseLink

Based on TopLink, contributed by Oracle, EclipseLink is an open source Eclipse persistence service that aims to provide a complete and pervasive persistence solution. It provides an extensible framework that allows Java developers to interact with various data services including databases, web services, OXM, enterprise information systems (EIS). EclipseLink supports some standards of the persistance API, including [4]`[U+FF1A]`(1) Object-Relational (JPA);(2) NoSQL (NoSQL Databases, and EIS); (3) Java Connector Architecture (JCA); (4) MOXy: Object-XML (JAXB) Object-JSON; (5) DBWS: Database Web Services; (6) Service Data Objects (SDO);

EclipseLink persistence platform consists of several components, such as EclipseLink-ORM, EclipseLink-OXM, EclipseLink-SDO, EclipseLink-DAS, EclipseLink DBWS, EclipseLink-XR, EclipseLink-EIS. Also, EclipseLink provides a complete JPA EJB3.0 compatible implementation. It offers full compatibility for all mandatory features, many optional features, and some extra features. Additional non-mandatory features include: object-level cache, distributed cache coordination, extensive performance tuning options, enhanced Oracle Database support, advanced mappings, optimistic and pessimistic locking options, extended annotations and query hints. EclipseLink provides support for EJB 3.0 container-based deployments, including web containers and other non-EJB 3.0 Java EE containers.

We can perform EclipseLink JPA's object-relational mapping using metadata annotations, using XML, overwriting and merging XML, Defaulting Properties, configuring entities, declaring basic attribute mappings, mappings, mapping inheritance, and using embedded objects.

- **Use meta annotations:** You can use annotations to configure the persistent behavior of the entity. For example, to specify a Java class as a JPA entity

- **Use XML:** We can use XML mapping metadata, metadata annotation, or you can overwrite metadata annotations.
- **Overwrite and merge XML:** You can use EclipseLink's native metadata xml file, EclipseLink-ORM.XML, to override the mapping of the orm.xml configuration file defined in JPA and provide the ORM feature for the EclipseLink extension. The EclipseLink-ORM.xml file defines object-relational mapping metadata, which is built from an existing orm.xml file, which makes it more intuitive, low-profile, and easy to overwrite.
- **Property default value:** Each annotation has a default value (refer to the JPA specification). The default for a persistence tool definition is for most applications. You only need to provide the value to override the default value. Therefore, it is not necessary to provide a configuration value, but an exception. This is called a configuration exception.
- **Configure the entity:** You can configure the identity of your entity, as well as the entity's phase-locked technology and sequence generation. Each entity must have a persistent identity, which is equivalent to the primary key of a database table to store the state of the entity.

3.3 Hibernate

"Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC)". [14]

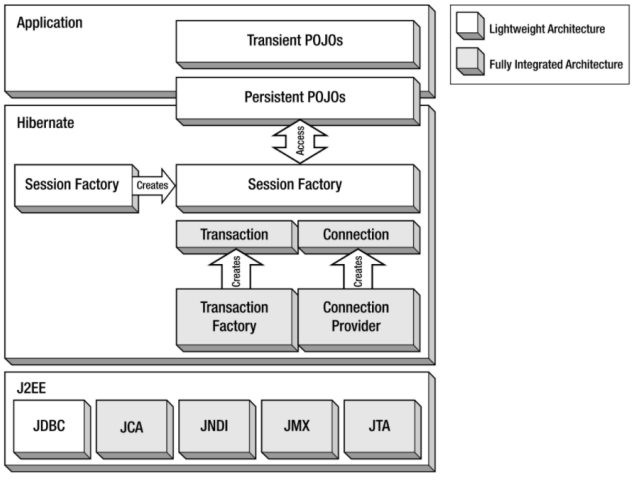


Figure 5: Hibernate Architecture. Source:[9]

As we can observe from Figure X, Hibernate Session Factory is one of the most important concepts of this ORM tool. The Section Factory is the responsible to receive the database information and manage it's connection retrieving the Hibernate Section. With the Hibernate Session is possible to perform SQL operations such as insert, update or deletes in a easy way, just like dealing with objects from Java classes. This classes should be mapped in a way that the Hibernate can understand and then translate to the appropriated query operations. When accessed inside the application the Hibernate Session object represent the bridge between application and database.

3.3.1 Hibernate Mappings

Hibernate also adopts Java Persistence API mappings syntax, the concepts of Hibernate and JPA mapping actually mix themselves in it's implementation once that some sources says that Java persistence API (JPA) was heavily influenced by Hibernate [5].

Hibernate offers to the developer two options of mapping: (1) mapping with annotations inside the classes, figure 6; (2) mapping using XML files and Hibernate Syntax, figure 7.

```java
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity

public class Example {
    @Id
    public Integer id;
    public String name;

    //constructor
    //...
    //getters and setters
}
```

Figure 6: Hibernate Annotation mapping example.

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE
  hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"
>

<hibernate-mapping default-access="field">
    <class name="Example">
        <id type="int" column="id">
            <generator class="native"/>
        </id>
        <property name="name" type="string"/>
    </class>
</hibernate-mapping>
```

Figure 7: Hibernate .hbm.xml mapping example

For the development of this study we will adopt only the .hbm.xml mapping as the default mapping for Hibernate use. Hibernate uses the mapping metadata to find out how to load and store objects of the persistent class.

The mapping file should be readable and hand-editable and the .hbm.xml file is Java-centric what in another words means that the XML mapping are build around the Java class persistence. The XML specifies each class to it's referee table as well as properties and columns its types and restrictions. Operations performed through HIbernate are simple and transparent to the developer. As said before this operations are made using Hibernate Session, the bridge between application and database, figure Z.

```
1  // Creates object
2  Example ex = new Example();
3  ex.setName("Advanced Database Project");
4  //Gets Hibernate session instance
5  Session session = HibernateUtil.openSession();
6  // Save the information on the db.
7  Transaction tx = session.beginTransaction();
8  session.save(ex);
9  tx.commit();
10 session.close();
```

Figure 8: Hibernate Operation Example

4 Use Case

In this chapter, a real example using object-relational database with Hibernate mapping tool will be presented. We will using a web application to show a scenario. The implementation will be in Java. The entity relationship model, database schema, as well as initial data and samples will be given in the case. And we will give some tests in order to see the performance of object relational database and hibernate mapping tool.

Scenario description As many students of BDMA are looking for accomodation in Barcelona, we are using the scenario of a house renting website to show the installation, application and performance of object-relational database.

The system has three different kinds of users:landlord, tenant, admin. For different users, it provides different functions.

For the landlord, he/she can register his/her basic information into the database, as well as the property information. All the information will be stored and shown to the tenants. When the landlord is logged in, he/she can check whether there is request of rent to his/her property. Once there is, the landlord could check the information of tenants then decide to approve or decline. He/she can upload information of more than one property. Also, one property could get more than one request. All the requests will be sent to the landlord and decided by him/her.

For the tenants, they can register their own personal information into the database and browse all the related property resource. The tenants could check the information of the property and the landlord to decide whether he/she wants to rent the property. Once they find some properties interesting, they could apply directly to the landlord. A tenant could send one or more applications to one or more landlords.

The admin has access to all the database. He/she could browse all the information of the landlords and their properties, as well as the tenants' information. Also, the admin has the access to every transaction to see whether there is anything abnormal. He/she also has to check the information of both sides to make sure that they are true and effective information.

At first, the process of direct contact between landlord and tenant is expected to happen outside of the web application. Only when submitting a lease request the tenant will get to see the landlord contact information, telephone and email. Same occurs for landlord, only when he/she receive the request the contact information of tenant will be available. Is expected that the landlord approve or decline tenant request after they have contacted each other. Once the request is approved, the property will be removed from the available properties period, until the lease finishes or landlord cancel it in the system.

4.1 Scenario Diagrams

As mentioned before the procedure is like the following:

1. First landlords register their personal information and post their proper-

ties online;

2. Tenants could see all suitable property resources after signing up and logging in, accessing all the details of the property in the database;
3. Once the tenants find suitable housing, he/she click the button to apply, system will send the request to the landlord, then the landlord could have access to the information of the tenant in the database;
4. The landlord will decide whether to approve or not;
5. Once the landlord approves the request, the property is removed from the available properties list until the end of the lease or lease cancellation by the landlord.

Based on the procedure above, the Entity Relationship Model was developed as following:

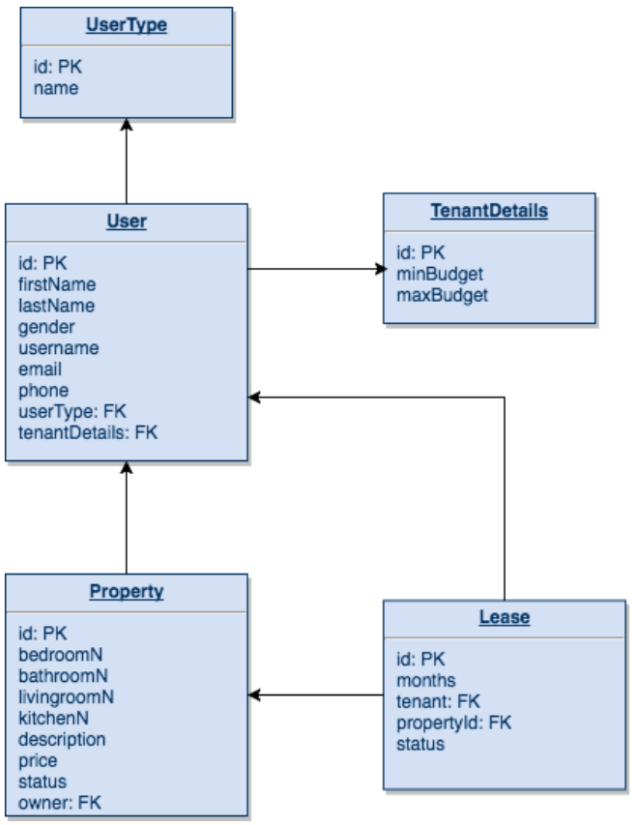


Figure 9: User Case ER Diagram

Also the Classes Diagram was elaborated to support the use case development using Java, Hibernate and Spring Framework.

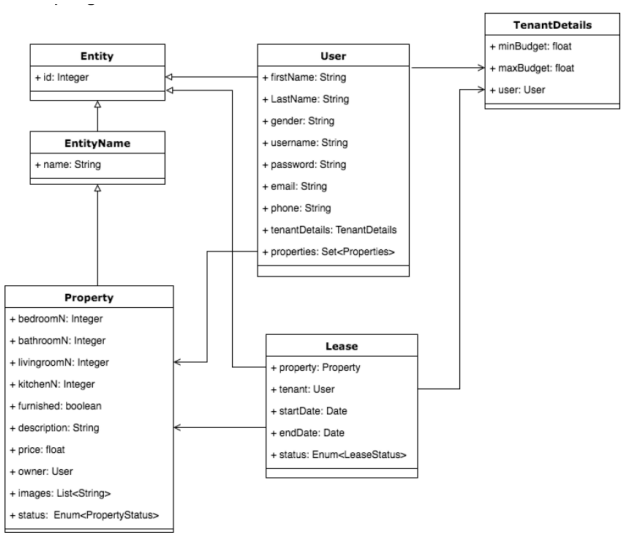


Figure 10: User Case Class Diagram

4.2 Implementation

To implement the described scenario, technologies such as Eclipse IDE, MySQL, Hibernate and Spring Framework were used. In this section, we want to focus on the use of the Hibernate ORM tool, showing its functionalities and power. As mentioned on section 4.1.4 Hibernate supports different types of mapping for Java classes (notations mapping and XML mapping). For this example, we make use of XML mapping, creating .hbm.xml files that correspond to the .java classes in the models package, figure 11.

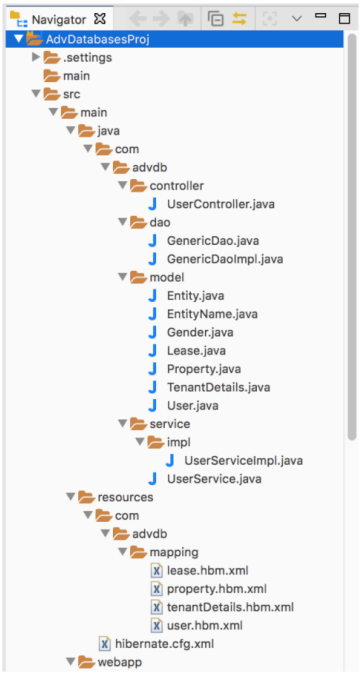


Figure 11: Project backend structure.

What we want with this scenario implementation is to show the usability of Hibernate and how much it can make the relationship developer / application / database different by means of simplicity. The project created was a Java Maven Web Project, what generates the structure showed on the previous picture. After the creation, the dependencies for Spring and Hibernate (figure x) were added on the pom.xml file. This make all the functionalities of theses 2 tools available in the whole scope of the project.

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.6.Final</version>
</dependency>
```

Figure 12: Hibernate dependency

The following images show the mapping of the classes created for the use case and its mappings in .hbm.xml. On these examples we are going to see different types of mapping and how they are handled by the Hibernate framework.

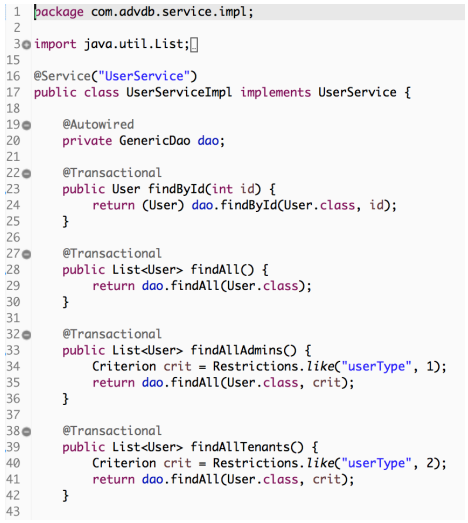
Also the Hibernate Operations where used in a generic class that has been reused for the necessary methods of the application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.advdb.model.User" table="users">
        <id name="id" type="int" column="id">
            <generator class="increment" />
        </id>
        <property name="firstName">
            <column name="firstName" not-null="true"/>
        </property>
        <property name="lastName">
            <column name="lastName" not-null="true"/>
        </property>

        <property name="gender">
             <column name="gender" not-null="true"/>
             <type name="org.hibernate.type.EnumType">
                 <param name="enumClass">com.advdb.model.Gender</param>
                 <param name="type">12</param>
                 <param name="useNamed">true</param>
             </type>
        </property>
        <property name="username">
            <column name="username" unique="true" not-null="true"/>
        </property>
        <property name="password">
            <column name="password" not-null="true"/>
        </property>
        <property name="email">
            <column name="email" unique="true"/>
        </property>
        <property name="phone">
            <column name="phone"/>
        </property>
        <property name="userType">
            <column name="userType" not-null="true"/>
        </property>
        <one-to-one name="tenantDetails" class="com.advdb.model.TenantDeta
             cascade="save-update"></one-to-one>
        <set name="properties" table="properties"
                inverse="true" lazy="true" fetch="select">
            <key>
                <column name="id" not-null="true" />
            </key>
            <one-to-many class="com.advdb.model.Property" />
        </set>
    </class>
</hibernate-mapping>
```

Figure 13: Hibernate User class mapping

```java
package com.advdb.dao;

import java.util.List;

@Repository
public class GenericDaoImpl implements GenericDao{

    @Autowired
    private SessionFactory sessionFactory;

    @SuppressWarnings("unchecked")
    public <T> T save(final T o){
        return (T) sessionFactory.getCurrentSession().save(o);
    }

    public void delete(final Object object){
        sessionFactory.getCurrentSession().delete(object);
    }

    /***/
    public <T> T findById(final Class<T> type, final int id){
        return (T) sessionFactory.getCurrentSession().get(type, id);
    }

    public <T> List<T> findAll(final Class<T> type, Criterion... criterions ) {
        Criteria crit = sessionFactory.getCurrentSession().createCriteria(type);
        for(int x=0; x<criterions.length; x++ ){
            crit.add(criterions[0]);
        }
        return crit.list();
    }

}
```

Figure 14: Hibernate Operations



```java
package com.advdb.service.impl;

import java.util.List;

@Service("UserService")
public class UserServiceImpl implements UserService {

    @Autowired
    private GenericDao dao;

    @Transactional
    public User findById(int id) {
        return (User) dao.findById(User.class, id);
    }

    @Transactional
    public List<User> findAll() {
        return dao.findAll(User.class);
    }

    @Transactional
    public List<User> findAllAdmins() {
        Criterion crit = Restrictions.like("userType", 1);
        return dao.findAll(User.class, crit);
    }

    @Transactional
    public List<User> findAllTenants() {
        Criterion crit = Restrictions.like("userType", 2);
        return dao.findAll(User.class, crit);
    }

```

Figure 15: Hibernate Operations Use

5 Conclusion

Hibernate makes the development of applications much more practical. Its easy syntax and the pre-defined handle operations makes it easier for the developer to code.

Hibernate mainly solves the object-relational impedance mismatch, the history of hibernate and ORM tools are very bounded together. Hibernate has huge influence over the paradigm as well as the JPA convection.

Also, for being there for many years Hibernate is the best bet for an ORM tool to be used with Java, once that it has been here in the market for several years now. In this way, Hibernate has a a big community out there as well a long source of resources trough all its versions.

The methods provided by Hibernate makes the CRUD operations less repetitive and all other Hibernates practices can make the difference in the application development time.

However, points made, Hibernate sometimes is understood as a villain when performing the operations over the database, once that its generated code can implies in performance lost over huge databases.

References

[1] *4 Benefits of Object-Relational Mapping (ORM) — Official Blog*. http://blogs.learnnowonline.com/2012/08/28/4-benefits-of-object-relational-mapping-orm/. (Accessed on 11/24/2017).

[2] *Apache OpenJPA EJB 3.0 [U+FF0C]1 OpenJPA EJB 3.0*. https://www.ibm.com/developerworks/cn/java/j-lo-openjpa1/index.html. (Accessed on 11/24/2017).

[3] Christian Bauer and Gavin King. *Java persistence with Hibernate: revised edition of Hibernate in action*. Manning, 2008.

[4] *EclipseLink[U+FF1A]JPA[U+3001]JAXB[U+3001]JCA SDO ORM*. http://hao.jobbole.com/eclipselink/. (Accessed on 12/20/2017).

[5] *Hibernate: save,persist, update, merge — Baeldung*. http://www.baeldung.com/hibernate-save-persist-update-merge-saveorupdate. (Accessed on 12/20/2017).

[6] *JPA Annotation OR Mapping*. https://www.ibm.com/developerworks/cn/java/j-lo-jpa-anntation/index.html. (Accessed on 11/24/2017).

[7] *Multi-database object-oriented proxy - Database Administrators Stack Exchange*. https://dba.stackexchange.com/questions/728/multi-database-object-oriented-proxy. (Accessed on 11/24/2017).

[8] *Object-relational database*. Dec. 2017. URL: https://en.wikipedia.org/wiki/Object-relational_database.

[9] Brian Sam-Bodden. *Beginning POJOs: from novice to professional*. Apress, 2006.

[10] Thomas Stoltmann. *Object-relational Mapping*. URL: https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/archive/ws1314/sp_semtext/07_ORM.pdf.

[11] *The Advantages of Object Relational Database — Techwalla.com*. https://www.techwalla.com/articles/the-advantages-of-object-relational-database. (Accessed on 11/24/2017).

[12] *What is an Object-Relational Database (ORD)? - Definition from Techopedia*. URL: https://www.techopedia.com/definition/8714/object-relational-database-ord.

[13] *What is Object-Relational Database Systems? Advantages and Disadvantages of ORDBMSS*. http://ecomputernotes.com/database-system/adv-database/object-relational-database-systems. (Accessed on 11/24/2017).

[14] *Your relational data. Objectively. - Hibernate ORM*. http://hibernate.org/orm/. (Accessed on 12/20/2017).

[15] Haseeb Yousaf. "Performance evaluation of Java Object-Relational Mapping tools". PhD thesis. 2012.