Enterprise systems

Behzad Bordbar
Lecture 10
Recap

• Sessions and its API
• Querying and sample of HQL
• Pagination
• Cascading
• Transactions- concurrency problems
  • Lost Updates
  • Unrepeatable Reads
  • Phantom Read
• Hibernate uses versioning
Long transactions

1) break the long transaction up into two (or more) database transactions

2) use detached objects from the first transaction to carry the necessary information to the presentation layer

3) Reattach the object to do necessary updates (as objects get modified by others)
Long transactions

Transaction management is a complex subject outside the scope of the course, but:

Safe solution:

1. read from the datatabase in the early transactions,
2. allow database writes in the last database transaction of a long transaction,
3. use versioning to check if your writes still make sense.
Versioning

Persistent object:
Add a version instance variable of type `int` or `long` to the objects.
Don’t forget accessor and mutator
Type can also be `TimeStamp` or `Calendar`

Advantage: you can see the time of the version

Disadvantage: timing problem (close updates will be treated as identical versions)
Mapping file `XX.bhm.xml`:

Update mapping file:

If the version attribute is called “version” add this after `<id>`

```xml
<version name="version"
    column="version">
```

Your database will have one extra column for versioning.
Versioning

Hibernate will take care of the versioning: whenever you make an object dirty in memory, Hibernate will update its version (in memory).

Hibernate throws

StaleObjectStateException

if the version number of the object on disk is not the same as it was when the object was loaded.
Versioning

Catch the exception and do something about the conflict

For example, report back to the user that the choice he/she has just made is, in fact, no longer available and could they please make another one
Objects to be persisted

Mapping files
X.hbm.xml, Y.bhm.xml
...

Configuration related
hibernate.properties
log4j.properties
ehcache.xml ...

Mapping-overall view
Hibernate objects

- **Entity beans**: objects with persistent identity
  (identifier field managed by Hibernate).
  Example: User, Customer, Order etc.

- **Value beans**: objects which only exist in relationship to an entity bean.
  Example: Address, CreditCardDetails etc.
Hibernate objects

• For each entity beans `sth.java` create a mapping file `sth.hbm.xml` to describe the mapping.

• Put `sth.hbm.xml` in the same directory as `sth.class`.

• The connection between any value beans related to `sth.java` is described in the mapping file of the corresponding entity bean, i.e. `sth.hbm.xml`
Simple Entity Classes without Relationships

Consider a class `User` with three attributes:
- `dateOfBirth: java.util.Date`
- `Username: String`
- `Gender: String` // or enum

We add `id` and `version` and all setters and getters to create an entity bean.
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
  "http://hibernate.sourceforge.net/hibernate-
  mapping-3.0.dtd">

<hibernate-mapping package="a.b.c">

<class name="User" table="user" lazy="true">
  <id name="id"
      column="id" type="long">
    <generator class="sequence"/>
  </id>

  <version name="version" column="version"/>
</class>
</hibernate-mapping>
Simple Entity Classes without Relationships

```xml
<property name="dateOfBirth" column="dob" type="date"/>
<property name="username" not-null="true" unique="true"/>
<property name="gender"/>
</class>
</hibernate-mapping>
```
Recap: what you have seen:

- Why ORM and Hibernate?
- A simple example to provide an overview
- Database patterns
- Log4j
- Object lifecycle in Hibernate
- Equality
- Session and its use
- Querying (some HQL!)
- Cascading persistence
- Transactions
- Mapping
Birds-eye view (RECAP)

Objects to be persisted

Runtime
- objX
- objY
- obj
- Main

Hibernate

Database

Mapping files
X.hbm.xml, Y.bhm.xml
...

Configuration related
- hibernate.properties
- log4j.properties
- ehcache.xml ...

132
When do we create hbm file? (RECAP)

- **Entity beans**: objects with persistent identity
  (identifier field managed by Hibernate).
  Example: User, Customer, Order etc.

- **Value beans**: objects which only exist in relationship to an entity bean.
  Example: Address, CreditCardDetails etc.
Mapping Value Objects within Entities

Consider the following:
What does this mean?
What is the black diamond shape?
What does the arrow mean?
Mapping Value Objects within Entities

Technically not different from Person class with an attribute Address

So we can include in `Person.hbm.xml`

```xml
<property name="address"
    type="string"/>
```

But what if Address has its attributes such as street, postcode…?

Use *components*
Mapping Value Objects within Entities

Instead of a property in `Person.hbm.xml` include a component with its own property items:

```xml
<component name="address"
    class="Address">
    <property name="street"
        column="user_street"/>
    <property name="postcode"
        column="user_postcode"/>
</component>
```
1) child objects are wholly owned by their parents: you cannot have two different parents.

2) A null child property is represented in the database by setting to null all the fields corresponding to the child object. Thus loading such a row will result in a parent object with a null child property, not a parent object with a child object whose properties are all null.

3) Not only can you have multiple components in a class, but one can have multiple components of the same (child) class in a class: such HomeAddress and WorkAddress.
Mapping Simple Entity Classes without Relationships

What does this mean?

Implement the following and save a few Customer and Account objects. You need to create java files and bhm.xml files.

For solution see part01.zip
Account object might be accessed by various applications, Modify `Account.java` and `Account.hbm.xml`.

Look at both files.

Solution:
part01_version_only_account_files.zip
Mapping Value Objects within Entities

What does this mean?

```
Customer
- customerName : string
- age : int

HomeAddress
- street : string
- postcode : string
- houseNo : int

workAddress
- street : string
- postcode : string
- officeNo : int
```
Exercise

Implement this model and test it by saving a Customer object and its two addresses.

[No sample code for this one- it is straightforward]
Mapping inheritance

What does this mean?

```
Address
- street : string
- postcode : string

HomeAddress
- houseNo : int

workAddress
- officeNo : int
```
Mapping inheritance

Use a single table
Each row of the table can hold an object of any type from the hierarchy.
One additional column called **discriminator** to distinguish between different types in hierarchy
You don’t need to add a new field called discriminator- it is for Hibernate book keeping
Mapping inheritance

```xml
<class name="Address" table="addresses"

discriminator-value="only-address">
    <id name="id" column="id" type="long">
        <generator class="sequence"/>
    </id>
    <discriminator column="subclass" type="string"/>
    <property name="streetName" column="street" type="string"/>
    <property name="postcode"/>

    <subclass name="HomeAddress"

discriminator-value="home-address">
        <property name="houseNo"/>
    </subclass>

... the rest
```
Mapping inheritance

Be careful with using “not null”
For a sample see part3.zip
Many-to-One, Unidirectional Associations

What does this mean?
What is the meaning of the arrow?
Many-to-One, Unidirectional Associations

Implement this model and write a small program to save the following students and lecturers

Student s1 = new Student("Jim", 123345);
Student s2 = new Student("John", 4321);
Lecturer l1 = new Lecturer("Achim");
s1.setSupervisedBy(l1);
s2.setSupervisedBy(l1);

Solution: see part4.zip

We will examine the code now.
<hibernate-mapping>
  <class name="Student" table="students" lazy="false">
    <id name="id" column="student_id">
      <generator class="sequence"/>
    </id>
    <property name="studentName" />
    <property name="studentID" />
    <many-to-one name="supervisedBy" cascade="save-update" column="supervisor"/>
  </class>
</hibernate-mapping>
Many-to-One, Unidirectional Associations

```xml
<hibernate-mapping>
  <class name="Lecturer" table="lecturers" lazy="false">
    <id name="id" column="lecturer_id">
      <generator class="sequence"/>
    </id>
    <property name="lecturersName" column="lecturers_name"/>
  </class>
</hibernate-mapping>
```
If no value assigned to `cascade`, changes will not be updated. For example, change to the Lecturer

- Restricting only the behaviour of `delete()`, `load()`... does not seem applicable
- `save-update` is convenient as it will reattach the Lecturer object if a lecturer object gets detached.
Modify the Main method of the previous example to add a new student who is supervised by a lecturer other than Achim, say Uday.

```java
Student s3 = new Student("Sue", 5467);
Lecturer l2 = new Lecturer("Uday");
s3.setSupervidedBy(l2);
```

Then write a new transaction to update the new student to be supervised by Achim.

**Sample solution:** see part5 of the zip file
What to do with 0..1 and *?

specify unique="true" as an attribute of the many-to-one element to have multiplicity * (no zero)

specifying not-null="true" to end up with multiplicity "1" instead of "0..1"

You can also take care of this by writing Assertions or using other constraint related libraries (OCL)- very much out of the scope of the course
One-to-Many, Unidirectional Associations

What does this mean?

`-hasAccount`

- `value : int`
- `IBAN : int`