# How is it possible that Hibernate works ?

Hibernate is an object/relational persistence and query service for Java, see http://www.hibernate.org . While the documentation, Hibernate' Wiki site and the preview on The Server Side "Chapter 6: Retrieving objects efficiently" teach very clearly how to *use* hibernate, I am quite curious on finding *how* actually Hibernate works; I vaguely know that "it is based on runtime reflection", and sibylline sentences like

> "This is how Hibernate implements what we call "runtime association fetching strategies," a powerful feature that is essential for achieving high performance in ORM."
> P. 19 of "Chapter 6: Retrieving objects efficiently"

increased my thirst. So I set up a simple example, and took notes in my experiments, which you find below.

I used screenshots from Intellij' Aurora java IDE ( http://www.intellij.com ). I assume familiarity with Hibernate' guide.

## 1 The example

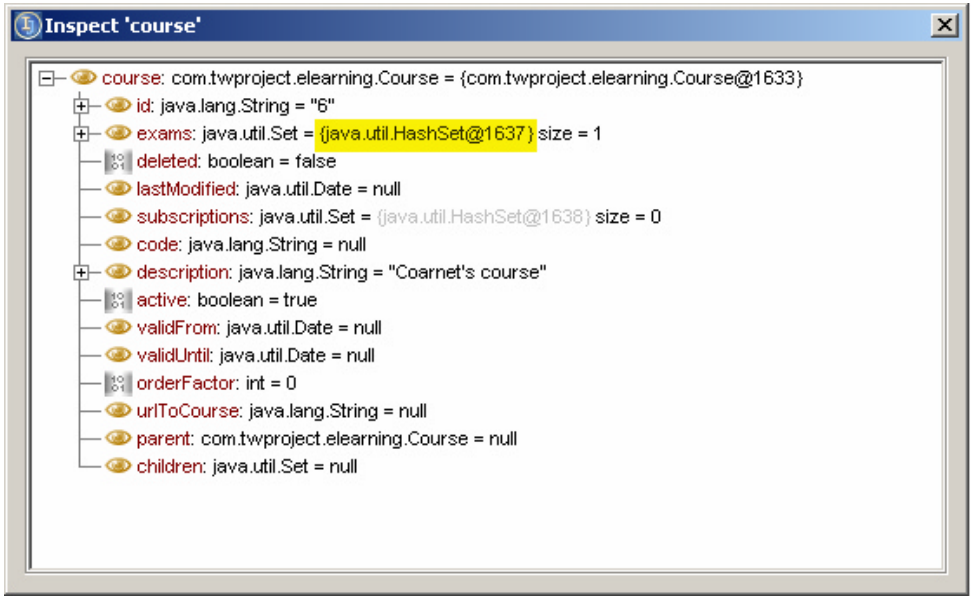
```
public class Course implements Securable, Comparable {

  private String id;
  private Set exams = new HashSet();
```

```
public class Exam implements Identifiable {

  private String id;
  private Set questions = new HashSet();
```

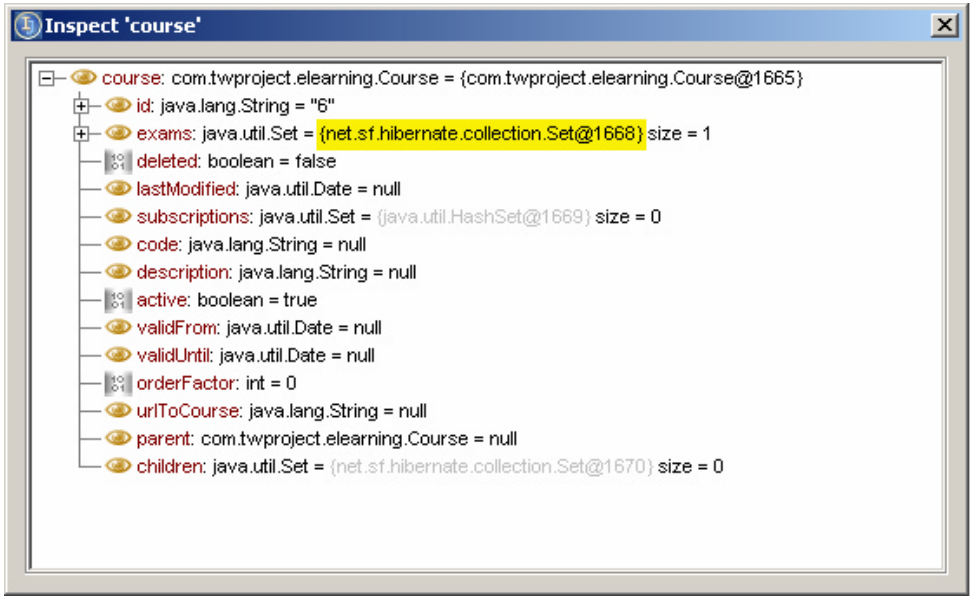
Suppose you create a Course sampleCourse, and set on it some exams; then you store the object, using Hibernate. Then you retrieve sampleCourse from the database, always through Hibernate.

These are the two objects you get:

before storing:




after retrieving from database:



See ? The two versions of course are different: the principle is

**once you store an object, it will never be the same**

This because via (possibly enhanced by CGLIB) reflection Hibernate inspects the object it is about to give you, and instantiates the exposed collections. Our business logic sees only the standard collections; but the wrapping permits interception of calls and management of behaviour, as we will see.

# 2 Working with collections

## 2.1 Code example

"So let me be very clear: Hibernate *completely* solves the n+1 SELECTs problem! However, it takes some thought and a (very) little work on the part of the user to take full advantage of this fact. We recommend that all associations be configured for lazy fetching by default. Then, for particular use cases, eager outer join fetching may be chosen by specifying a LEFT JOIN FETCH clause in a HQL query, or by calling setFetchMode() for a Criteria query."
Gavin King' blog
http://blog.hibernate.org/cgi-bin/blosxom.cgi/2004/01/27

One of the main advantages of using Hibernate is the approach to lazyness, and the number of options given by the application and its query language.

Our code goes:

```
public class Test {

  public static void main(String[] args) {

    Transaction t = null;
    try {
      t = TransactionHome.getInstance().create();

      OqlQuery oq = new OqlQuery("from " + Course.class.getName() + " course where course.id = :id");
      oq.getQuery().setString("id", "1");
      Course course = (Course) oq.uniqueResult();
```

I step over, and hence instantiate my Course. How this is instantiated depends on my mappings, hence we consider various cases.

## 2.2 Simplest case

```
<class name="com.twproject.elearning.Course" table="el_crs">

  <id column="id" name="id" type="string" length="50">
    <generator class="assigned"/>
  </id>

  <set name="exams" table="el_crs_exm">
    <key column="course_id"/>
    <many-to-many class="com.twproject.elearning.Exam" column="ex_id" />
  </set>
```

As our Course has an exam in exams, this is the resulting hql/sql:

find: from com.twproject.elearning.Course course where course.id = :id
named parameters: {id=1}

HQL: from com.twproject.elearning.Course course where course.id = :id
SQL: select course0_.id as id, … where (course0_.id=? )
about to open: 0 open PreparedStatements, 0 open ResultSets

select course0_.id as id, … where (course0_.id=? ) [gets the course]

select exams0_.ex_id as ex_id__, exams0_.course_id as course_id__ from el_crs_exm exams0_ where exams0_.course_id=? [get the course exam collection]

select exam0_.id as id0_, … where exam0_.id=? [get its first exam]
select exam0_.id as id0_, … where exam0_.id=? [get its second exam]

Fours selects are done 1 for the course + 1 for the collection + 2 for each record, sorting a bit, 1 + 2 + 1 i.e. 1 + (n=number of records) + 1 . In this case it seems peachy, but could quickly become a nightmare, as we wanted only our course, but we got all that was related to it. Here Hibernate wrapping features come to help.

## 2.3 Use a lazy collection

We modify slightly the mapping:

```
<class name="com.twproject.elearning.Course" table="el_crs">

  <id column="id" name="id" type="string" length="50">
    <generator class="assigned"/>
  </id>

  <set name="exams" table="el_crs_exm" lazy="true">
    <key column="course_id"/>
    <many-to-many class="com.twproject.elearning.Exam" column="ex_id" />
  </set>

 ...
```

Notice the lazy=”true” on the set of exams on the course.

```
find: from com.twproject.elearning.Course course where course.id = :id
named parameters: {id=1}

HQL: from com.twproject.elearning.Course course where course.id = :id

SQL: select course0_.id as id, … where (course0_.id=? )

resolving associations for [com.twproject.elearning.Course#1]
creating collection wrapper:[com.twproject.elearning.Course.exams#1]
creating collection wrapper:[com.twproject.elearning.Course.children#1]
creating collection wrapper:[com.twproject.elearning.Course.subscriptions#1]
done materializing entity [com.twproject.elearning.Course#1]
```

Only *one* select is done. The log above shows only a part of what Hibernate does; for example, Hibernate checks whether it is hitting the session (transaction) cache, and (if present) the second level cache; so keep in mind that here we are seeing only one of the means to achieve efficiency.

Now, we got only primitive (plus String and Date) properties of our Course, but what happens if I want the collection ? Do I have to explicitly tell Hibernate this, so making my application dependent and fragile ? No, see how nicely this is handled: suppose we execute line 27

```
27        Set exams = course.getExams();
28        for (Iterator iterator = exams.iterator(); iterator.hasNext();) {
29          Exam exam = (Exam) iterator.next();
30        }
```

Now, as we have seen in the above sql, before line 27-29 course.getExams() is *not* initialized with the data on the database; but the idea is that Course, being a persistent class, has all its persistent collections, in our case a Set, initialized as a net.sf.hibernate.collection, in our case a net.sf.hibernate.collection.Set :

```
public class Set extends ODMGCollection implements java.util.Set …
```

so the call to iterator.next(), where instantiation is unavoidable, triggers the following calls, as the Hibernate collection is persistence context aware:

```
initializing collection [com.twproject.elearning.Course.exams#1]

select exams0_.ex_id as ex_id__, exams0_.course_id as course_id__ from el_crs_exm exams0_ where exams0_.course_id=?
```

select exam0_.id as id0_, … where exam0_.id=?

Three selects: one to retrieve the collection the other two to retrieve its elements (there are two exams). If the lazy attribute is not used, as in this case, no further calls are needed apart from the initial three.

Notice that the collection elements are initialised if you use iterator; but if they have collections on them, and these are marked as "lazy", these will not be initialized in our simple loop, but only if needed. If you want to get data about the collection that does not require even the primitive properties, there are ways to do this in hql, e.g.

```
Set exams = course.getExams();
int size = ((Integer)session.createFilter(exams,"select count(*)").iterate().next()).intValue();
```

This will issue one query whatever the size of the collection.

But there is still lurking the

**dreaded n+1 problem**
To retrieve a n-sized collection by n+1 sql queries, one for the collection members ids, and then 1 for each member, instead of 1

The solution is this:

```
25        OqlQuery oq = new OqlQuery(
26            "from " + Course.class.getName() + " course " +
27            "left join fetch course.exams " +
28            "where course.id = :id");
29        oq.getQuery().setString("id", "1");
30
31        Course course = (Course) oq.uniqueResult();
32
33        Set exams = course.getExams();
34        for (Iterator iterator = exams.iterator(); iterator.hasNext();) {
35          Exam exam = (Exam) iterator.next();
36          System.out.println(exam.getId());
37        }
```

At line 31, the following is issued:

HQL: from com.twproject.elearning.Course course left join fetch course.exams where course.id = :id

SQL: select course0_.id as id0_, exam2_.id as id1_, course0_.parent as parent0_, exams1_.ex_id as ex_id__, exams1_.course_id as course_id__ from el_crs course0_ left outer join el_crs_exm exams1_ on course0_.id=exams1_.course_id left outer join el_exa exam2_ on exams1_.ex_id=exam2_.id where (course0_.id=? )

No further queries are needed for the following. Hence indeed the dreaded n+1 problem is solved, actually we tranformed n+1 additional queries to zero !

Another approach: suppose that exams are not “cascading lazyness”, so to retrieve them one query is never sufficient. In this case let Hibernate supply us with an iterator:

```
38        Query oq = session.createQuery(
39            "select exam from " + Exam.class.getName() + " exam, " +
40            Course.class.getName() + " course " +
41            "where exam in elements(course.exams) and course.id = :id");
42        oq.setString("id", "1");
43        Iterator iter = oq.iterate();
44        while (iter.hasNext()) {
45          Exam exam = (Exam) iter.next();
46          System.out.println(exam.getId());
47        }
```

By executing line 43, only one sql is executed:

> HQL: select exam from com.twproject.elearning.Exam exam, com.twproject.elearning.Course course where exam in elements(course.exams) and course.id = :id
>
> SQL: select exam0_.id as x0_0_ from el_exa exam0_, el_crs course1_ where (exam0_.id in(select exams2_.ex_id from el_crs_exm exams2_ where course1_.id=exams2_.course_id))and(course1_.id=? )

Then the exams will be retrieved on need; this is different from above where we brutally did course.getExams(), hence forcing the initialisation of all of them. In the example above, we iterate through all the exams, but of course this may not be necessary.

## 2.4 The parent/child case

Our course is actually a case of a parent/child relationship, as we modeled a course' lesson as a course. This is mapped like this, following Hibernate documentation:

```
<class name="com.twproject.elearning.Course" table="el_crs">

  <id column="id" name="id" type="string" length="50">
    <generator class="assigned"/>
  </id>

  <set name="exams" table="el_crs_exm" lazy="true">
    <key column="course_id"/>
    <many-to-many class="com.twproject.elearning.Exam" column="ex_id" />
  </set>

  <many-to-one name="parent"/>
  <set name="children"                _          inverse="true" lazy="true">
    <key column="parent"/>
    <one-to-many class="com.twproject.elearning.Course"/>
  </set>
```

Notice as usual the lazy="true" of the children. Hence with the code:

```
25          Course course = (Course) oq.uniqueResult();
26
27          Set children = course.getChildren();
28          for (Iterator iterator = children.iterator(); iterator.hasNext();) {
29            Course child = (Course) iterator.next();
30            System.out.println(child.getDescription());
31          }
```

at step 28 *one* query is issued, and that's all:

initializing collection [com.twproject.elearning.Course.children#1]
select children0_.id as id__, … where children0_.parent=?

## 2.5 Many and one to one associations

For writing this section I was helped by http://www.hibernate.org/162.html .

We have seen how to handle the collections case, by using collection wrappers. Now change the mapping as follows:

```
<class name="com.twproject.elearning.Course" table="el_crs">

  <id column="id" name="id" type="string" length="50">
    <generator class="assigned"/>
  </id>

  <one-to-one name="exam" />
```

Now if we call course.getExam(), Hibernate has no way to intercept this, as Exam is defined by us. So in this situation, exam must be fetched. So for the code

```
28          OqlQuery oq = new OqlQuery(
29              "from " + Course.class.getName() + " course " +
30              "where course.id = :id");
31          oq.getQuery().setString("id", "6");
32
33          Course course = (Course) oq.uniqueResult();
34          Exam exam = course.getExam();
```

at step 33 we get the following sql calls:

```
find: from com.twproject.elearning.Course course where course.id = :id
named parameters: {id=1}

HQL: from com.twproject.elearning.Course course where course.id = :id

SQL: select course0_.id as id, course0_.exam as exam, course0_.parent as parent from el_crs course0_ where (course0_.id=? )
select course0_.id as id, course0_.exam as exam, course0_.parent as parent from el_crs course0_ where (course0_.id=? )

select exam0_.id as id0 … where exam0_.id=?

select questions0_.qs_id as qs_id__, … where questions0_.ex_id=?
```

So also exam has been fetched, together with its non lazy collections. At step 34 we have no need of further queries.

So now

**how do we give Hibernate control over many and one to one relations ?**

**by introducing proxies**

The guide:

> "Hibernate implements lazy initializing proxies for persistent objects using runtime bytecode enhancement (via the excellent CGLIB library)."
> 12.1 of Hibernate' guide

A proxy as defined in

> Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns*, Addison Wesley, 1994

is a class that acts as a stand in for the real class, permitting real instantiation on demand. This is exactly what we needed.

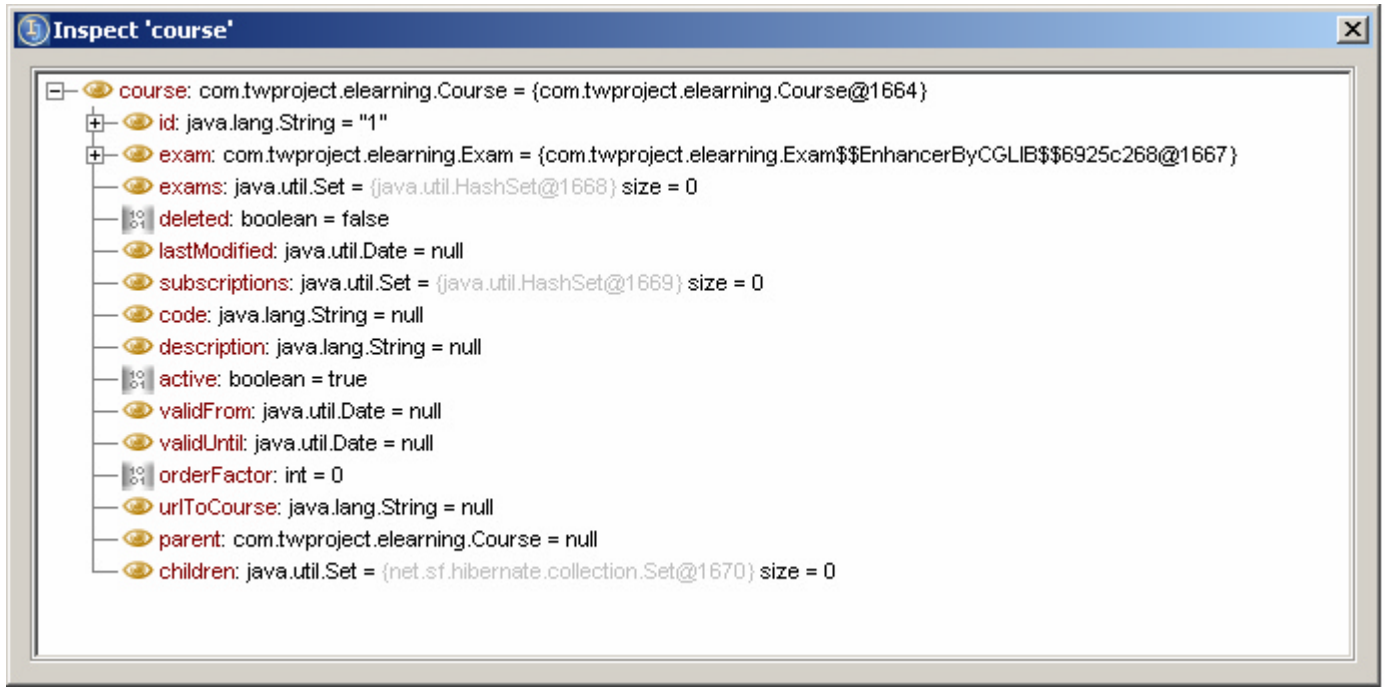
So modify the mapping by

```
<class name="com.twproject.elearning.Exam" table="el_exa" proxy="com.twproject.elearning.Exam" >

  <id column="id" name="id" type="string" length="50">
    <generator class="assigned"/>
  </id>

  ...
```

It may look strange to use the same class as a proxy of itself; it is a way to authorize the framework to use a proxy.

HQL: from com.twproject.elearning.Course course where course.id = :id
SQL: select course0_.id as id, course0_.exam as exam, course0_.parent as parent from el_crs course0_ where (course0_.id=? )

Now only one query is done. Inspect the course object:




See ? The exam is wrapped by a proxy created on the fly by using runtime reflection. And only when the exam is really needed, it is instantiated:

```
select exam0_.id as id0_, … where exam0_.id=?
select questions0_.qs_id … where questions0_.ex_id=?
```

# 3 Conclusion

I hope I have transmitted one of the core ideas that make Hibernate work. I am happy because this way I understood much more of its internal workings, which helps to write efficient code. Feedback at ppolsinelli@open-lab.com would be great.

**Disclaimer**
Nothing here reported shold be read as contradicting/overwriting what stated in the official Hibernate documentation: in such cases, rely on the official documentation. Here I do not claim ownership of any trademark whatsoever.

**Author**
Pietro Polsinelli is a founder of open lab, a dynamic Italian company developing software solutions, and a java enthusiast. His most beloved java child is teamwork, http://www.twproject.com , which of course uses Hibernate as its persistence layer.