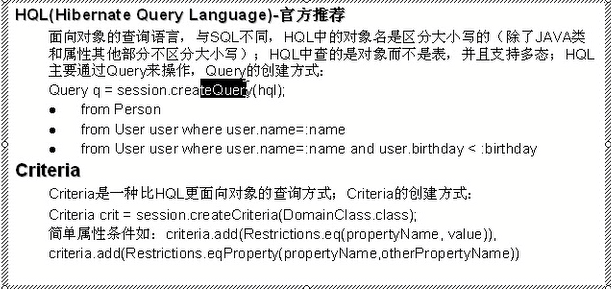
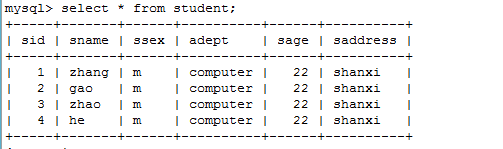
# HQL 是面向对象的查询语句



# 下面通过一个学生信息系统进行管理

# 1.创建相应数据库

## 1.1建立学生表student



create table student(

sid Integer primary key auto\_increment,

sname varchar(20) not null,

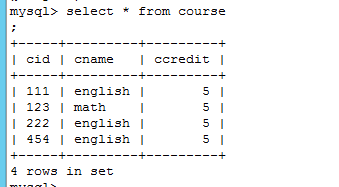
ssex varchar(2) not null,

adept varchar(20) not null, 系

sage Integer not null,

saddress varchar(50) not null);

## 2.建立课程表course



create table course(

cid Integer primary key,

cname varchar(20) not null,

ccredit Integer not null);

# 3.建立选课表stuCourse

### 注意的是，这个选课表中含有外键,必须在主表中存在

create table stuCourse(

stuCourseId Integer auto\_increment primary key,

sid Integer not null,

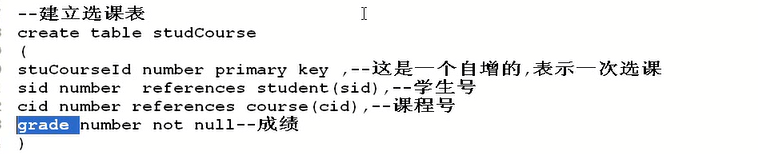
cid Integer not null,

foreign key(sid) references student(sid),

foreign key(cid) references course(cid),

grade Integer not null);

insert into stucourse(sid,cid,grade) values(3,111,96);



# 4.如果我们的表有主外键的关系，应该先映射主表，在映射从表

## 4.1我们的学生表student中出现了下面的东西，就是因为他里面有数据被当做了外键,

**private** Set stucourses = **new** HashSet(0);

## 4.2选课表Stucourse的变化最大，外键直接变成对象了，因为我们的主外键关系要体现出来，外键从表的角度看是关系模型，关系模型移植到对象模型，有自己的一套方案

**public** **class** Stucourse **implements** java.io.Serializable {

**private** Integer stuCourseId;

**private** Student student;

**private** Course course;

**private** Integer grade;

## 4.3学生表中的集合的

//通过这个集合表明一个学生可以选择多门课程，反过来一门课程可以被多次选择

**private** Set stucourses = **new** HashSet(0);

## 4.4打开Stucourse.hbm.xml 中，我们发现有了<many-to-one多对一

<many-to-one name=*"student"* class=*"com.hlj.domain.Student"* fetch=*"select"*>

<column name=*"sid"* not-null=*"true"* />

</many-to-one>

<many-to-one name=*"course"* class=*"com.hlj.domain.Course"* fetch=*"select"*>

<column name=*"cid"* not-null=*"true"* />

</many-to-one>

## 4.5而学生表中是一对多，就是下面就是体现了一个集合了

<set name=*"stucourses"* inverse=*"true"*>

<key>

<column name=*"sid"* not-null=*"true"* />

</key>

<one-to-many class=*"com.hlj.domain.Stucourse"* />

</set>

## 4.6下次讲解关系映射



# 5.然后我们就开始进行测试HQL语句了

## 5.1检索所有学生的信息

//检索所有学生的信息

**private** **static** **void** selectAll(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Query query=session.createQuery("from Student");

List<Student> list=query.list();

**for**(Student e: list){

System.*out*.println("姓名"+e.getSname()+"性别"+e.getSsex());

}

//使用iterator迭代器进行输出

Iterator<Student> iterator = list.iterator();

**while** (iterator.hasNext()) {

Student student = iterator.next();

System.*out*.println("iterator姓名"+student.getSname()+"性别"+student.getSsex());

}

ts.commit();

} **catch** (Exception e) {

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage()+"这个错误了");

}**finally**{

//关闭session

**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}

## 5.2检索部分学生的in西

//检索学生的名字和有限字段

//原则上我们是 可以像jdbc中全部查找出来的

//hibernate 建议中都是必须都查出来

//但是我们也要学会 查找有限的

**private** **static** **void** selectOne(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Query query=session.createQuery("select sname,adept from Student");

//这样我们就不可以使用下面的东西了，就会报错因为哦我们不能得到一个list

List list=query.list();

**for**(**int** i = 0 ; i < list.size();i++){

Object []object = (Object[]) list.get(i);

System.*out*.println(object[0].toString()+""+object[1]);

}

Iterator iterator = list.iterator();

**while** (iterator.hasNext()) {

Object[] objects = (Object[]) iterator.next();

System.*out*.println(objects[0].toString()+"fadsf"+objects[1]+"fasdf");

}

ts.commit();

} **catch** (Exception e) {

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage()+"这个错误了");

}**finally**{

//关闭session

**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}

# 6使用Student中的set语句

//通过这个集合表明一个学生可以选择多门课程，反过来一门课程可以被多次选择

**private** Set stucourses = **new** HashSet(0);

## 6.1学生选择了几门课程

**private** **static** **void** StudentSet(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Query query=session.createQuery("from Student");

//这样我们就不可以使用下面的东西了，就会报错因为哦我们不能得到一个list

List<Student> list=query.list();

**for**(Student s: list){

System.*out*.println(s.getSname()+"选择了"+"课程表: "+

s.getStucourses()+"并且选择了"+

s.getStucourses().size()+"门课程");

}

ts.commit();

} **catch** (Exception e) {

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage()+"这个错误了");

}**finally**{

//关闭session

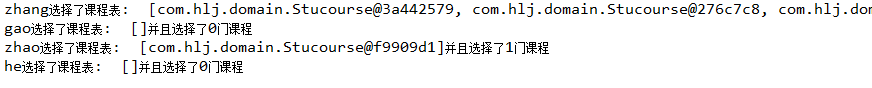
**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}



## 6.2直接找学生选择了几门课程

**private** **static** **void** StudentgetCourse(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Query query=session.createQuery("from Student");

//这样我们就不可以使用下面的东西了，就会报错因为哦我们不能得到一个list

List<Student> list=query.list();

**for**(Student s: list){

**if**(s.getStucourses().size()==0){

System.*out*.println(s.getSname()+"没有选课");

}**else**{

//System.out.println(s.getSname()+"选择了"+"课程表: "+s.getStucourses()+"并且选择了"+s.getStucourses().size()+"门课程");

Set<Stucourse> stucourses = s.getStucourses();

**for**(Stucourse stc : stucourses){

System.*out*.println(s.getSname()+"选择了"+stc.getCourse().getCname());

}

}

}

ts.commit();

} **catch** (Exception e) {

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage()+"这个错误了");

}**finally**{

//关闭session

**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}

# 7.uniqueResult();确认有一条数据，我们就是用

//确认只有一条记录的时候使用 uniqueResule

**private** **static** **void** uniqueResule(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Student student = (Student)session.createQuery("from Student where sid = '1'").uniqueResult();

System.*out*.println(student.getSname());

ts.commit();

} **catch** (Exception e) {

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage()+"这个错误了");

}**finally**{

//关闭session

**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}

# 8. distinct学生的性别和年龄，但是不能出现重复

**private** **static** **void** selectSexAndAge(){

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts = session.beginTransaction();

//检索所有学生的信息

Query query=session.createQuery("

select distinct ssex,sage from Student");

//这样我们就不可以使用下面的东西了，就会报错因为哦我们不能得到一个list

List list=query.list();

**for**(**int** i = 0 ; i < list.size();i++){

Object []object = (Object[]) list.get(i);

System.*out*.println(object[0].toString()+""+object[1]);

}

# 9.范围查找20到23岁之间的

Query query=session.createQuery("from Student

where sage between 20 and 23");

# 10查找包含某一类型的

List<Student> list=session.createQuery

("from Student where adept in ('computer')").list();

# 11.count 必须要分组

## 11.1查找系里面的平均年龄

//下面这个是两个字段了 ，系里面的平均年龄

List<Object[]> list=session.createQuery("select avg(sage),adept from Student group by adept").list();

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" "+obj[1].toString());

}

## 11.2. //查询各个系有 人数 大于3 的系的名称 //查询这个系里有多少个学生

//count 用于别名 as c1

List<Object[]> list=session.createQuery("select count(\*) as c1,adept from Student group by adept having count(\*)>3").list();

/

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" "+obj[1].toString());

}

## 11.3//选择系里面男生有多少个 ，和系的名称

List<Object[]> list=session.

createQuery("select count(\*) as c1,adept from Student where ssex='m' group by adept").list();

//ȡ��1. for ��ǿ

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" "+obj[1].toString());

}

14

# 12查询一个字段 ，obj.toString().如果还是像以前一样会报错的

List<Object[]> list=session.

createQuery("select sage from Student where adept='computer'").list();

//ȡ��1. for ��ǿ

**for**(Object obj:list){

System.*out*.println(obj.toString());

}

# 13主外键进行查找，

## 13.1查找课程为111的最大成绩和最小成绩

List<Object[]> list=session.

createQuery("select 111,max(grade),min(grade) from Stucourse where course.cid=111").list();

//ȡ��1. for ��ǿ

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" max="+obj[1].toString()+" min="+obj[2].toString());

}

## 13.2 //显示各科考试不合格的成绩和学生

List<Object[]> list=session.

createQuery("select student.sname,course.cname,grade from Stucourse where grade>=60").list();

//ȡ��1. for ��ǿ

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" "+obj[1].toString()+" "+obj[2].toString());

}

## 13.3 选择成绩 大于60的 学生的数目和所在的系

List<Object[]> list=session.

createQuery("select count(\*),student.adept from Stucourse where grade>60 group by student.adept").list();

//ȡ��1. for ��ǿ

**for**(Object[] obj:list){

System.*out*.println(obj[0].toString()+" "+obj[1].toString());

}

15

# 14.分页

**private** **static** **void** showResultByPage(**int** pageSize){

//���÷�ҳ�ı���

**int** pageNow=1;

**int** pageCount=1;//����

**int** rowCount=1;//�����Ҫ��ѯ

Session session=**null**;

Transaction tx=**null**;

**try** {

session=HibernateUtil.*getCurrentSession*();

tx=session.beginTransaction();

//��ѯ��rowcount

rowCount=Integer.*parseInt*(session.createQuery("select count(\*) from Student").uniqueResult().toString());

pageCount=(rowCount-1)/pageSize+1;

//�������ǿ���ѭ������ʾÿҳ����Ϣ

**for**(**int** i=1;i<=pageCount;i++){

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*第"+i+"页\*\*\*\*\*\*\*\*\*\*\*\*");

//设置第几条结果开始 然后 乘以 每页的大小 得到的集合

List<Student> list=session.createQuery("from Student").setFirstResult((i-1)\*pageSize)

.setMaxResults(pageSize).list();

**for**(Student s: list){

System.*out*.println(s.getSname()+" "+s.getAdept());

}

}

tx.commit();

}**catch**(Exception e){

e.printStackTrace();

**if**(tx!=**null**){

tx.rollback();

}

**throw** **new** RuntimeException(e.getMessage());

}**finally**{

**if**(session!=**null**&&session.isOpen()){

session.close();

}

}

}