Hibernate Queries

- Introduction to HQL
- HQL Syntax

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Queries

- Most applications need to work with sets of entities
- Hibernate provides several ways to query entities:
    - The Hibernate Query Language (HQL)
    - The Hibernate Query by Criteria (QBC) and Query by Example (QBE) API
    - The Java Persistence Query Language (JPQL)
    - Native SQL
- Native SQL lets you use database-specific syntax at the expense of portability

---

This chapter covers only HQL.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Query Language History

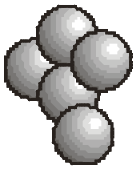
- Hibernate has always defined HQL; QBC and QBE are included in Hibernate version 3
- The JPQL is based on HQL, with syntax borrowed from the EJB Query Language
- To use JPQL with Hibernate, your project must include the **Hibernate Entity Manager** and **Hibernate Annotations**

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL and JPQL vs Standard SQL

- The object query languages work with entity objects and their properties, not tables and columns
- The object query languages are portable across databases, while native SQL varies from database to database

Object Space





Relational Space

STUDENT

STUDENTID	NAME	DATE
5	Jess Settles	1998-12-02
6	George Peeples	1966-05-19
7	Ben McGilmer	1965-03-13

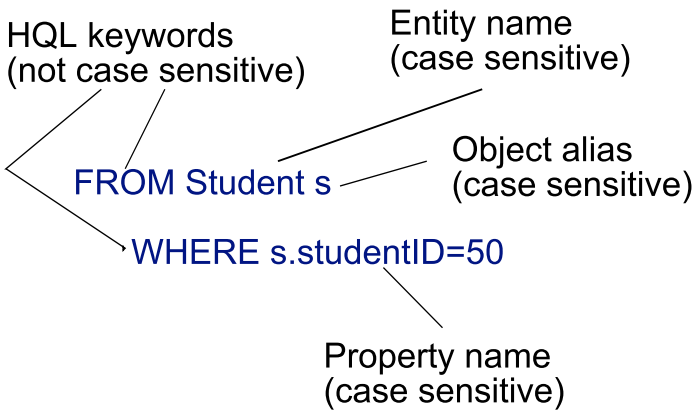
To be portable, HQL doesn't use SQL syntax. Instead, it lets you query totally in object space. The persistence manager then converts the portable HQL into database-specific SQL.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic HQL Query Syntax

- HQL queries work with entity class names and their property names
- Unlike in standard SQL, class and property names **are** case-sensitive

```
public class Student
{
  private int studentID;
  private String name;
  private double gpa;
. . .
}
```



Essentially, using this abstract schema lets you construct queries totally in object space.

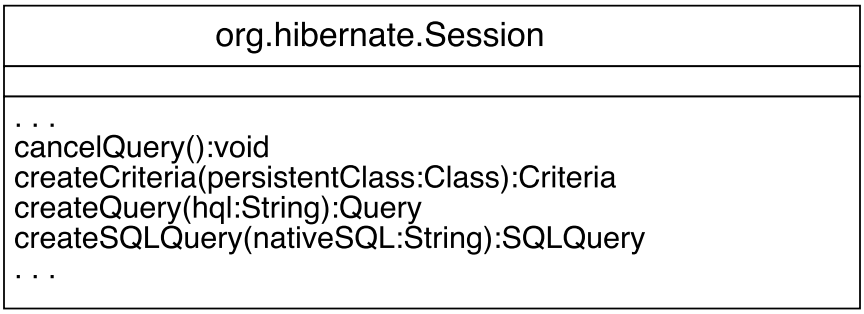
Note that unlike in standard SQL, entity and property names are case sensitive.

Native queries use actual table names and columns and the persistence manager assumes that the entity property and class names exactly match.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating an HQL Query

- Before you can execute a query, you must create an **org.hibernate.Query** from the Hibernate session




```
Query q = session.createQuery("FROM Student");
```


The Session has several other responsibilities, including persisting objects and providing a transaction reference.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Query Interface

org.hibernate.Query
. . . executeUpdate():int list():List setBigDecimal(name:String, bd:BigDecimal):Query setBigInteger(name:String, bi:BigInteger):Query setBoolean(name:String, b:boolean):Query setByte(name:String, b:byte):Query setCalendar(name:String, cal:Calendar):Query setCharacter(name:String, ch:char):Query setDate(name:String, date:Date):Query setDouble(name:String, d:double):Query setEntity(name:String, val:Object):Query setFetchSize(size:int):Query setFirstResult(first:int):Query setFloat(arg0:String, arg1:float):Query setFlushMode(mode:FlushMode):Query setInteger(name:String, i:int):Query setLong(name:String, l:long):Query setMaxResults(max:int):Query setParameter(name:String, val:Object):Query setReadOnly(b:boolean):Query setShort(name:String, s:short):Query setString(name:String, s:String):Query setTime(name:String, dt:Date):Query setTimeout(timeout:int):Query setTimestamp(name:String, dt:Date):Query uniqueResult():Object . . .

```
Query q =
  session.createQuery(
   "FROM Student");

List<Student> students;

students = q1.list();
```


We will cover many of these methods in this chapter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Single Results vs Multiple Results

- Some queries return a single result: an object or a simple value
- Some queries return multiple objects or values

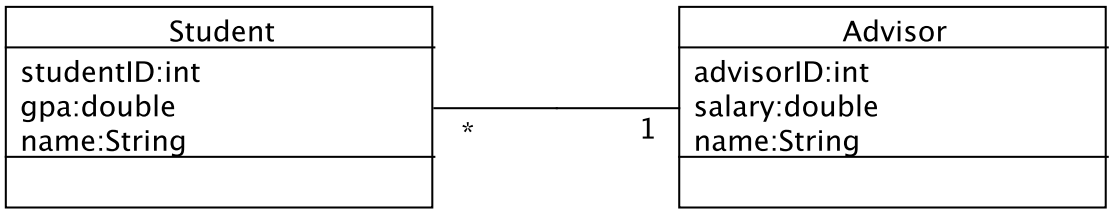
```
1    Query q1 = session.createQuery(
2       "FROM Student s WHERE s.studentID=2");
3    Student s = (Student)q1.uniqueResult();
4
5    Query q2 = session.createQuery("FROM Student");
6    List<Student> students = q2.list();
```


The first query specifies the Student's unique primary key, so that query will return at most one object. If there are no results, the query will return null. If by chance the query actually returned multiple entities, uniqueResult() throws the unchecked org.hibernate.NonUniqueResultException.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Entities for Queries

- The examples that follow will use these two entities
- Note the bidirectional one-to-many relationship




```
1    Query q1 = session.createQuery(
2      "FROM Student s WHERE s.studentID=2");
3    Student s = (Student) q1.uniqueResult();
4
5    Query q2 = session.createQuery("FROM Advisor");
6    List<Advisor> advisors = q2.list();
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Query Parameters

- Like JDBC PreparedStatements, HQL queries can accept runtime parameters
- Unlike JDBC, parameters can be either named or positional

```
1    Query q = session.createQuery(
2      "FROM Student s WHERE s.studentID=:id");
3    q.setParameter("id", 2);
4
5    Student s = (Student) q.uniqueResult();
```

Instead of calling setParameter(), you can instead call setXXXXX() where XXXXX is the parameter type. The setParameter() call attempts to determine the type of the parameter, whereas the setXXXXX() methods let you specify the type unambiguously.

JDBC PreparedStatements only allow positional parameters.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Query Syntax

- SELECT queries let you retrieve entities while optionally filtering, grouping and ordering the results

```
select_statement :: = select_clause from_clause
  [where_clause] [groupby_clause]
  [orderby_clause]
```

```
1    Query q1 = session.createQuery("FROM Student");
2    List<Student> students = q1.list();
3
4    Query q2 = session.createQuery(
5      "SELECT s.name FROM Student s");
6    List<String> names = q2.list();
```

Here we show a couple of examples of the SELECT and FROM clauses. In the first example, we note that the SELECT syntax itself is optional for simple queries.

In the second example, we specify SELECT and a property from the entity – the query will retrieve only the property values from all entities that match the query. Note that we must use the "s" alias at the end of the SELECT clause so we can refer to the property.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL Operators

- The WHERE clause lets you restrict the scope of the query
- HQL provides **operators** to let you specify which objects you wish to retrieve

Category	Operators
Navigation	.
Arithmetic	+,-,*,/
Logical	NOT, AND, OR
Comparison	=,<>,<,>,<=,>=, LIKE, BETWEEN, IN, IS NULL, IS EMPTY, MEMBER OF

HQL supports a few more operators, include parenthesis for grouping expressions.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL Operator Examples

```
// find students whose ID is NOT 52
FROM Student s
 WHERE s.studentID <> 52

// find highly paid advisors
SELECT a FROM Advisor a
  WHERE a.salary > 35000

// find moderately paid advisors
SELECT a FROM Advisor AS a
 WHERE a.salary BETWEEN 15000 AND 50000

// find highly taxed advisors with low IDs
SELECT a FROM Advisor a
  WHERE (a.salary * 0.1) > 7500
  AND a.advisorID < 5
```

The 'AS' keyword is optional.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL Functions

```
String Manipulation
-------------------
LOWER(string)
UPPER(string)
TRIM([LEADING | TRAILING | BOTH] string)
CONCAT(string1, string2)
LENGTH(string)
LOCATE(substr, string)
SUBSTRING(string, start, length)

Math
----
ABS(number)
SQRT(double)
MOD(int, int)
```


HQL defines a few other functions not listed here.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL Functions, cont'd

```
Returning Date/Time
-------------------
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP

SELECT Query Aggregates
-----------------------
COUNT(identifier)
MAX(identifier)
MIN(identifier)
AVG(numeric identifier)
SUM(numeric identifier)
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HQL Function Examples

```
// find count of advisors
SELECT COUNT(a) FROM Advisor AS a

// find average advisor salary
SELECT AVG(a.salary) FROM Advisor AS a

// find advisors with long names
// that contain 'Sue'
SELECT a FROM Advisor AS a
  WHERE LENGTH(a.name) > 6 AND
  LOCATE('Sue', a.name) <> 0
```


In HQL, the first index in a string is one (1), so the LOCATE function returns zero if the specified string is not found.

The functions that return a number return either a Double or a Long object. You can use code like this:

```
Query q = session.createQuery(
  "SELECT AVG(A.salary) FROM Advisor AS A");
Double avg = (Double)q.uniqueResult();
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Ordering

- The ORDER BY clause lets you control the order in which query results are returned
- By default, it sorts in ascending order but you can use the DESC keyword for descending

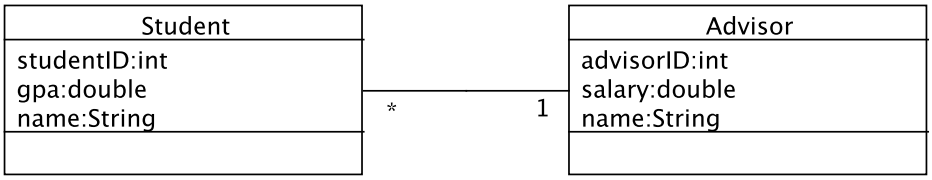
```
// find advisors sorted by salary
SELECT a FROM Advisor AS a
  ORDER BY a.salary

// find advisors sorted by salary
// descending order
SELECT a FROM Advisor AS a
  ORDER BY a.salary DESC
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Navigating Relationships

- You can use the INNER JOIN syntax to select related objects




```
// find all students for advisor with ID=1
SELECT s FROM Advisor AS a
  INNER JOIN a.students AS s
  WHERE a.advisorID=1
```


In this example, the Advisor's OneToMany collection is named "students" -- thus the "a.students" syntax above.

HQL also supports "LEFT JOIN" or "LEFT OUTER JOIN" that enable the retrieval of a set of entities where matching values in the join condition may be absent. For values that don't exist, the persistence manager returns null.

HQL also supports FETCH joins, which prefetches related objects even if the relationship is configured as a LAZY relationship. FETCH joins can help avoid the lazy instantation exception covered in the Relationships chapter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Subqueries

- Subqueries let you execute queries within queries

```
// determine how many advisors have above average salary
SELECT COUNT(a) FROM Advisor AS a
  WHERE a.salary >
    (SELECT AVG(x.salary) FROM Advisor AS x)
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Paging

- Paging is the act of repeatedly fetching a fixed number of a large result set for displaying on a page (e.g. a Web page)

```
1    public List<Student> getStudents(Session session,
2       int max, int index)
3    {
4        Query q = session.createQuery(
5          "SELECT s FROM Student s");
6        q.setFirstResult(index);
7        q.setMaxResults(max);
8        return q.list();
9    }
```


Here we start a code listing showing a method that queries all Students using no WHERE clause. But the query will not return ALL students: only those within the indexes specified by q.setFirstResult() and q.setMaxResults().

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Paging, cont'd

```
1    int first = 0;
2    int max = 10;   // entries per page
3    int page = 1;
4
5    List<Student> students =
6       getStudents(session, max, first);
7    while (students.size() > 0)
8    {
9       System.out.println("Displaying page " + page++);
10      for(Student s : students)
11      {
12         System.out.println("Name: " + s.getName());
13         System.out.println("ID  : " + s.getStudentID());
14      }
15
16      session.clear();
17      first += students.size();
18      students = getStudents(session, max, first);
19   }
```

Continuing the listing from the last page, here we repeatedly call the method shown on the last page, displaying a "page" of students at a time (we've defined a page to be 10 students).

The session.clear() call is not strictly required, but it does cause the session to mark the previous page of Students as detached, so they will be available for garbage collection.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned

- Fundamental concepts in querying sets of entities in object space
- How to compose various types of queries using the Hibernate Query Language

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum