Hibernate Query

Hibernate Query Capabilities (in 3.0)

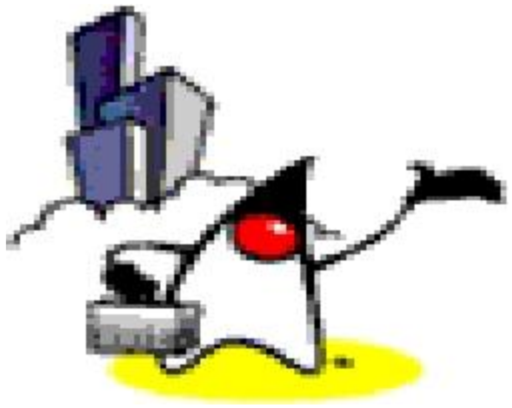
- Newly enhanced Hibernate Criteria Query API
- Hibernate Query Language (HQL)
- Enhanced support for queries expressed in the native SQL dialect of the database

Hibernate Criteria API

Topics

- What is Criteria query?
- How to use Criteria query API?
- Pagination
- Restrictions
- Ordering
- Aggregate functions
- Fetch modes
- Query By Example (QBE)

What is Criteria Query?



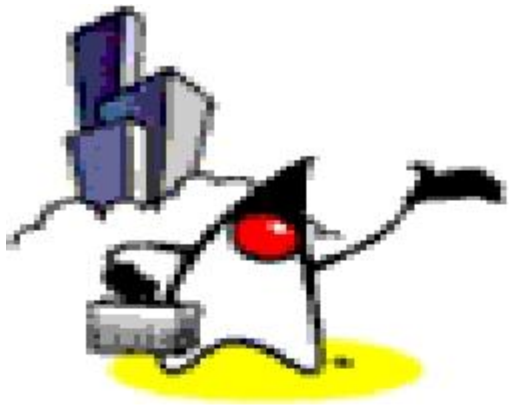
Three ways of retrieving data in Hibernate

- Criteria query API
  - The easiest way to retrieve data
  - Pure Java language based
- Hibernate Query Language (HQL)
- Native SQL query

Criteria Query API

- Uses a set of Java objects for constructing queries
  - Instead of query language
- Lets you build nested, structured query expressions in Java programming language
  - Compile time syntax checking possible
  - Polymorphic behavior – get instances of X & subclass(X)
- Supports Query By Example (QBE)
  - Performing a query by providing an example object that contain properties that need to be retrieved
- Supports aggregation methods (from Hibernate 3)
  - Count

How to use Criteria Query API

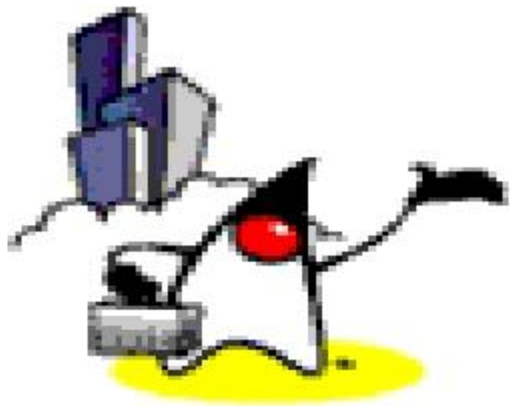


How to use Criteria Query API

- Create *org.hibernate.Criteria* object via *createCriteria()* factory method of the *Session*
  - Pass persistent object's class or its entity name to the *createCriteria*() method
- Call *list*() method of the *Criteria* object

```
// Get all instances of Person class and its subclasses
Criteria crit = sess.createCriteria(Person.class);
List results = crit.list();
```

Pagination

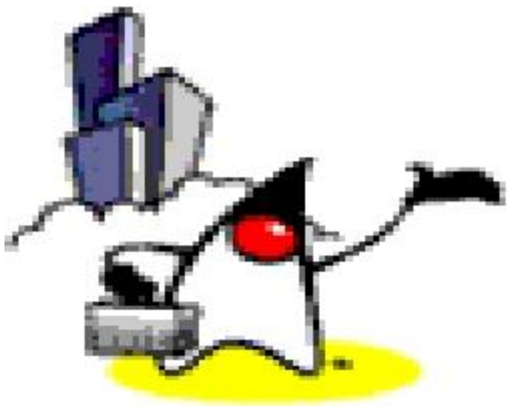


Pagination through the Result Set

- Hibernate handles the pagination
  - Retrieving fixed number of objects
- Two methods of Criteria class
  - *setFirstResult()* - set the first row in the result
  - *setMaxResults()* - number of rows to retrieve

```
Criteria crit = sess.createCriteria(Person.class);
crit.setFirstResult(2);
crit.setMaxResults(50);
List results = crit.list();
```

Narrowing the Result Set via Restrictions



Restrictions class

- Used to selectively retrieve objects
  - Example: Person objects whose age is over 20
- Add restrictions to the *Criteria* query object with *add()* method
  - The *add()* method of the *Criteria* object takes an *org.hibernate.criterion.Criterion* object that represents an individual restriction
- You can have more than one restriction for a Criteria query

Methods of Restrictions class

- *Restrictions.eq("name", "Shin")*
- *Restrictions.ne("name", "NoName")*
- *Restrictions.like("name", "Sa%")*
- *Restrictions.ilike("name", "sa%")*
- *Restrictions.isNull("name");*
- *Restrictions.gt("price",new Double(30.0))*
- *Restrictions.between("age", new Integer(2), new Integer(10))*
- *Restrictions.or(criterion1, criterion2)*
- *Restrictions.disjunction()*

Add a restriction

- *Restrictions.like()* - pattern based restriction

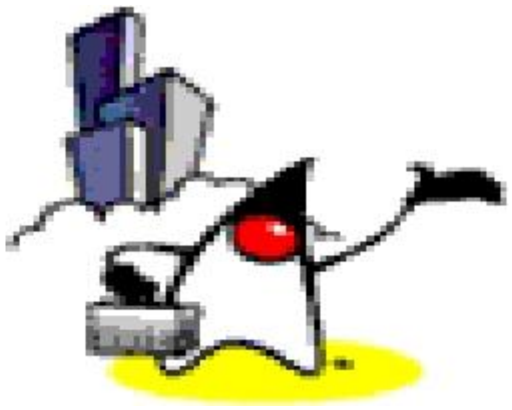
```
// Retrieve person objects whose name has a pattern
Criteria crit = sess.createCriteria(Person.class);
Criterion nameRestriction = Restrictions.like("name",
    "Shin%");
crit.add( nameRestriction );
List results = crit.list();
```

Logical Grouping of Restrictions

- Restrictions can be logically grouped

```
// Retrieve Person objects whose name has a pattern
// and whose age is 10 or null
List people = sess.createCriteria(Person.class)
    .add( Restrictions.like("name", "Shin%") )
    .add( Restrictions.or(
        Restrictions.eq( "age", new Integer(10) ),
        Restrictions.isNull("age")
    ))
    .list();
```

Ordering the Result Set

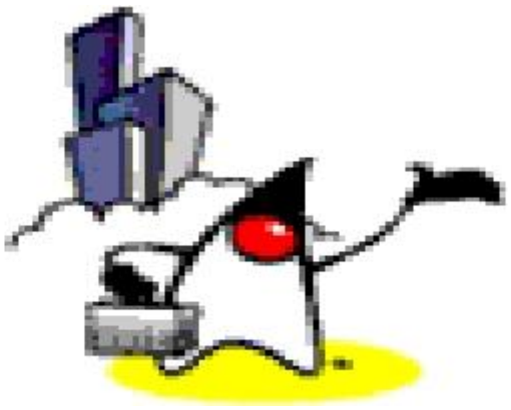


Ordering the results

- You may order the results using *org.hibernate.criterion.Order*

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(50)
    .list();
```

Projections & Aggregates



Aggregate functions available through Projections factory class

- *rowCount()*
- *avg(String propertyName)*
  - average of a property's value
- *count(String propertyName)*
  - number of times a property has a value
- *countDistinct(String propertyName)*
  - number of unique values the property contains
- *max(String propertyName)*
- *min(String propertyName)*
- *sum(String propertyName)*
  - sum of the property values

Projections

- Projections.rowCount()

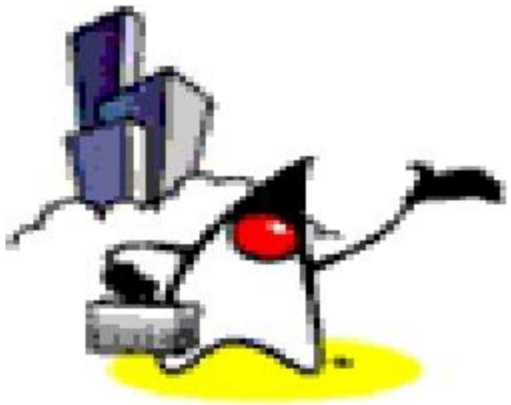
```
// The result will contain one object, an Integer that
// contains the results of executing COUNT SQL
// statement
Criteria crit = sess.createCriteria(Person.class);
crit.setProjection( Projections.rowCount() );
List results = crit.list();
```

Multiple Projections

- Projections.projectionList()

```
// You will get a List with an Object array
// as the first element. The Object array
// contains all the values in order
Criteria crit = sess.createCriteria(Product.class);
ProjectionList projectList = Projections.projectionList();
projectList.add(Projections.avg(“price”));
projectList.add(Projections.sum(“price”));
crit.setProjection( projectList );
List results = crit.list();
```

Fetch Modes



Fetching Modes (How it is fetched)

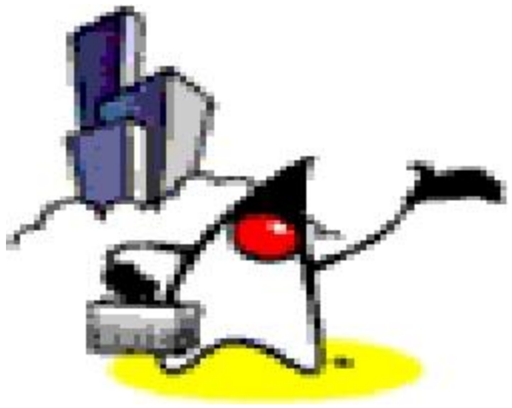
- FetchMode.DEFAULT
  - Default to the setting configured in the mapping file.
- FetchMode.JOIN
  - Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.
- FetchMode.SELECT
  - A second SELECT is used to retrieve the associated entity or collection.
  - Unless you explicitly disable lazy fetching by specifying *lazy="false"*, this second select will only be executed when you actually access the association.

Setting the Fetch Mode

- *setFetchMode("permissions", FetchMode.JOIN)*

```
User user = (User) session.createCriteria(User.class)
                .setFetchMode("permissions", FetchMode.JOIN)
                .add( Restrictions.idEq(userId) )
                .uniqueResult();
```

Query By Example (QBE)



What is Query By Example (QBE)?

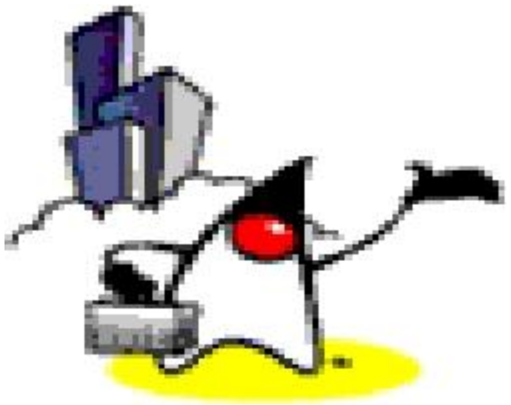
- Provides another style of searching
- How to perform QBE based query
  - Partially populate an instance of an object
  - Let Hibernate to build a criteria using the instance as an example behind the scene
- *org.hibernate.criterion.Example* class implements *Criterion* interface
  - You can use it like any other restrictions

Query By Example

- Use *Example.create()* to create a restriction

```
// Retrieve person objects via example object
Criteria crit = sess.createCriteria(Person.class);
Person person = new Person();
person.setName(“Shin”);
Example exampleRestriction = Example.create(person);
crit.add( exampleRestriction );
List results = crit.list();
```

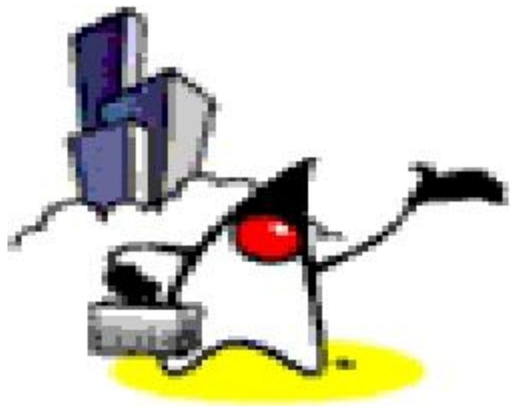
Hibernate Query Language (HQL)



Topics

- What is HQL?
- “from” clause
- Associations and join
- “select” clause
- Polymorphic query
- “where” clause

What is HQL?



Hibernate Query Language (HQL)

- Very similar to SQL but less verbose
- Understands OO – inheritance, polymorphism, association, aggregation, composition
  - Selection: *from, as*
  - Associations and joins: *inner join, outer join, right*
  - *outer join, full join*
  - Projection: *select, elements*
  - Constraints: *where*
  - Other constructs: *aggregate functions, expressions,*
  - *order by clauses, group by clauses, polymorphic*
  - *selections, sub-queries*

Why to use HQL?

- **Full support for relational operations**
  - HQL allows representing SQL queries in the form of objects. Hibernate Query Language uses Classes and properties instead of tables and columns.
- **Return result as Object:**
  - The HQL queries return the query result(s) in the form of object(s), which is easy to use. This elemenates the need of creating the object and populate the data from result set.
- **Polymorphic Queries:** HQL fully supports **polymorphic queries**. Polymorphic queries results the query results along with all the child objects if any.
- **Easy to Learn:** Hibernate Queries are easy to learn and it can be easily implemented in the applications.
- **Support for Advance features:** HQL contains many advance features such as pagination, fetch join with dynamic profiling, Inner/outer/full joins, Cartesian products. It also supports Projection, Aggregation (max, avg) and grouping, Ordering, Sub queries and SQL function calls.
- **Database independent:** Queries written in HQL are database independent (If database supports the underlying feature).

HQL

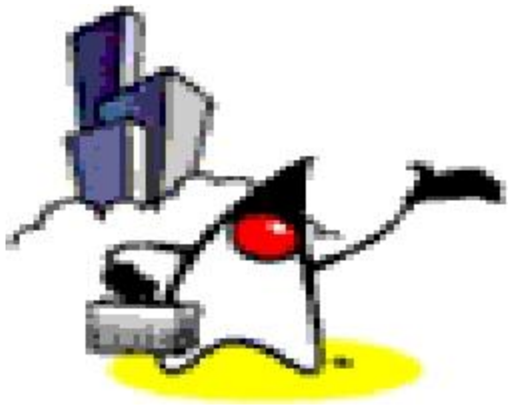
- **Clauses in the HQL are:**
  - **from**
  - **select**
  - **where**
  - **order by**
  - **group by**
- **Aggregate functions are:**
  - **avg(...)**, **sum(...)**, **min(...)**, **max(...)**
  - **count(*)**
  - count(...), count(distinct ...), count(all...)
- **Subqueries**

  It is a query within another query. Hibernate supports Subqueries if the underlying database supports it.

Differences from SQL

- HQL is fully object-oriented, understanding notions like inheritance, polymorphism and association
- Queries are case-insensitive, except for names of Java classes and properties
- Hibernate automatically generates the sql query and execute it against underlying database if HQL is used in the application.
- HQL is based on the relational object models and makes the SQL object oriented.
- Hibernate Query Language is used to execute queries against database.
- Hibernate Query Language uses Classes and properties instead of tables and columns.

“from” clause



“from” clause

- Return all instances of the class eg.Cat.

  from eg.Cat

- Usually don't need to qualify the class name, since auto-import is the default.

  from Cat is same as *from eg.Cat*

- Most of the time, you will need to assign an alias, since you will want to refer to the Cat in other parts of the query

  from Cat as cat (“cat” is the alias)

  from Cat cat (“as” can be omitted)

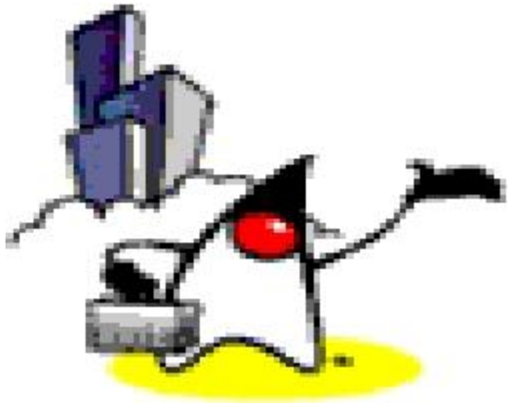
Alias

- Multiple classes may appear, resulting in a Cartesian product or "cross" join.

  ```
  from Formula, Parameter
  from Formula as form, Parameter as param
  ```

- Name query aliases using an initial lowercase, consistent with Java naming standards for local variables

Associations and joins



join

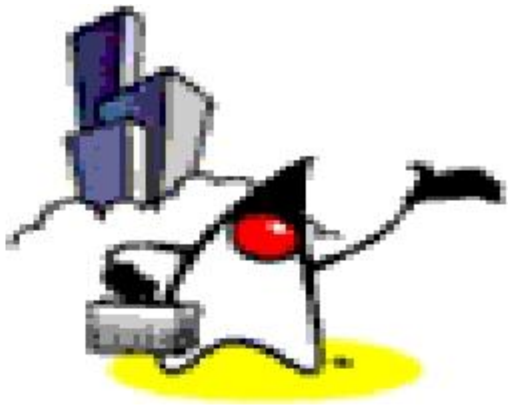
- We may also assign aliases to associated entities, or even to elements of a collection of values, using a join

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
from Cat as cat left join cat.mate.kittens as kittens
from Formula form full join form.parameter param
```

join types supported

- inner join
- left outer join
- right outer join
- full join (not usually useful)

“select” clause



select clause

- The select clause picks which objects and properties to return in the query result set

```
select mate
from Cat as cat
    inner join cat.mate as mate
```

- Compact form

```
select cat.mate from Cat cat
```

select clause

- Queries may return properties of any value type including properties of component type

```
select cat.name from DomesticCat cat
                    where cat.name like 'fri%'
select cust.name.firstName from Customer as cust
```

- Queries may return multiple objects and/or properties as an array of type Object[]

```
select mother, offspr, mate.name
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

select clause

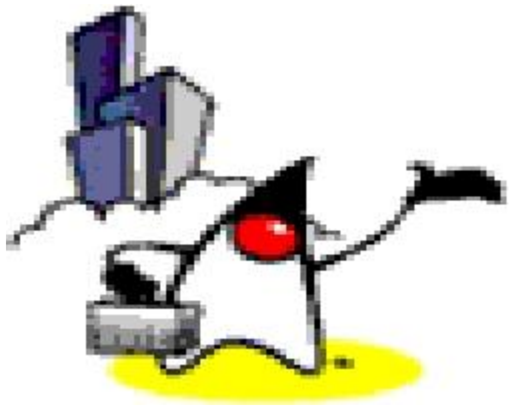
- Queries may return multiple objects and/or properties as a List

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

- Or as an actual typesafe Java object

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
    join mother.mate as mate
    left join mother.kittens as offspr
```

Polymorphic Query



Polymorphic queries

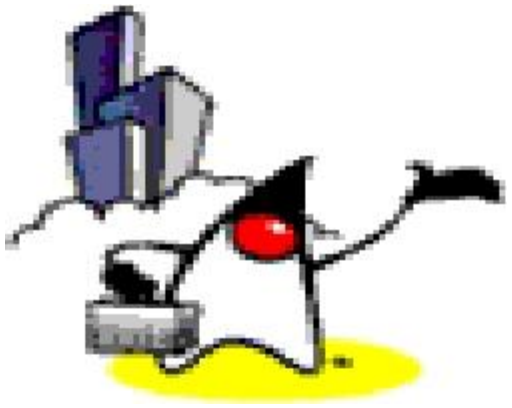
- The query below returns instances not only of Cat, but also of subclasses like DomesticCat

```
from Cat as cat
```

- Hibernate queries may name any Java class or interface in the from clause.
- The query will return instances of all persistent classes that extend that class or implement the interface.
- The following query would return all persistent objects

```
from java.lang.Object o
```

"where" clause



where clause

- The where clause allows you to narrow the list of instances returned.
- If no alias exists, you may refer to properties by name

```
from Cat where name='Fritz'
```

- If there is an alias, use a qualified property name:

```
from Cat as cat where cat.name='Fritz'
```

where clause

- Return all instances of Foo for which there exists an instance of bar with a date property equal to the startDate property of the Foo

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

- Compound path expressions make the where clause extremely powerful.

```
from Cat cat where cat.mate.name is not null
```