Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Hibernate

Searches and Queries

Lesson Objectives

- **In this lesson, you will learn:**
  - Hibernate Query Language(HQL)
  - Hibernate Query Language(HQL) Performance
    - Lazy Loading
    - Bulk updates
  - Query by criteria
  - Hibernate Projections

Hibernate Query Language(HQL)

- **HQL is a language for talking about “sets of objects”**
- **It unifies relational operations with object models**
- **HQL is an object-oriented query language, similar to SQL, but instead of operating on tables and**
- **columns, HQL works with persistent objects and their properties.**
- **HQL is a language with its own syntax and grammar.**

Hibernate Query Language

- **Hibernate Query Language is used to execute queries against database.**
- **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 uses Classes and properties instead of tables and columns.**
- **Hibernate Query Language is extremely powerful and it supports Polymorphism, Associations, Much less verbose than SQL**

Hibernate Query Language

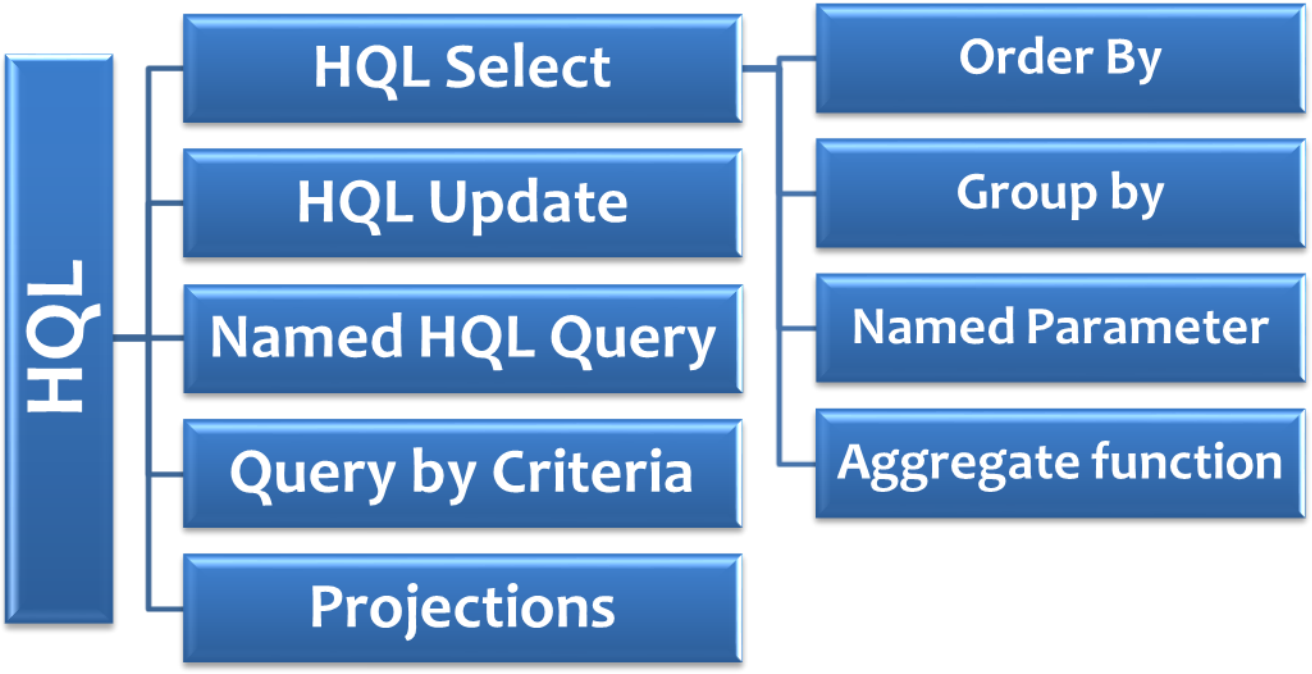
- **Make SQL be object oriented**
  - Classes and properties instead of tables and columns
  - Polymorphism
  - Associations
  - Much less verbose than SQL
- **Full support for relational operations**
  - Inner/outer/full joins, cartesian products
  - Projection
  - Aggregation (max, avg) and grouping
  - Ordering
  - Subqueries
  - SQL function calls

Why to use HQL?

- **Full support for relational operations**
- **Return result as Object**
- **Polymorphic Queries**
  - Polymorphic queries results the query results along with all the child objects if any
- **Easy to Learn**
- **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
- **Database independent**

Why to use HQL?

The following HQL queries are of significance and will be discussed:



The from Clause and Aliases

- **Simple shortcut query to select all objects from the Product table**

  “from Product as p or from Product as product”
- **The from clause is very basic and useful for working directly with objects.**
- **String SQL_QUERY ="from Insurance insurance";**

```
Query query = session.createQuery(SQL_QUERY);
for(Iterator it=query.iterate();it.hasNext();){
Insurance insurance=(Insurance)it.next();
System.out.println("ID: " +insurance.getLngInsuranceId());
System.out.println("First
Name: " + insurance.getInsuranceName());
} session.close();
```

HQL select Clause

- **The select clause provides more control over the result set than the from clause. If you want to obtain the properties of objects in the result set, use the select clause**

- **//Create Select Clause HQL**

```
String SQL_QUERY ="Select insurance.lngInsuranceId,
        insurance.insuranceName," + "insurance.investementAmount,
        insurance.investementDate from Insurance insurance";
Query query = session.createQuery(SQL_QUERY);
for(Iterator it=query.iterate();it.hasNext();){
Object[] row = (Object[]) it.next();
System.out.println("ID: " + row[0]);
System.out.println("Name: " + row[1]);
System.out.println("Amount: " + row[2]);
}session.close();
```

HQL select Clause

- **Where Clause is used to limit the results returned from database. It can be used with aliases and if the aliases are not present in the Query, the properties can be referred by name.**
- **String SQL_QUERY =" from Insurance**
  as insurance where insurance.lngInsuranceId='1'";
  Query query = session.createQuery(SQL_QUERY);
  for(Iterator it=query.iterate() ;it.hasNext();)
- **{**
  Insurance insurance=(Insurance)it.next();
  System.out.println("ID: " + insurance.getLngInsuranceId());
  System.out.println("Name: " +insurance.getInsuranceName());
  }

The where clause

- **The where clause allows you to refine the list of instances returned. If no alias exists, you can refer to properties by name:**
  - from Cat where name='Fritz'
  - from Cat as cat where cat.name='Fritz'
    - This returns instances of Cat named 'Fritz'.
  - select foo from Foo foo, Bar bar where foo.startDate = bar.date
    - returns all instances of Foo with an instance of bar with a date property equal to the startDate property of the Foo

The order by clause

- **The list returned by a query can be ordered by any property of a returned class or components:**
- **“from DomesticCat cat order by cat.name asc, cat.weight desc, cat.birthdate”**

- **The optional asc or desc indicate ascending or descending order respectively.**

The group by clause

- **A query that returns aggregate values can be grouped by any property of a returned class or components:**
  - select cat.color, sum(cat.weight), count(cat) from Cat cat group by cat.color .
- **A having clause is also allowed.**
  - select cat.color, sum(cat.weight), count(cat) from Cat cat group by cat.color having cat.color in (eg.Color.TABBY, eg.Color.BLACK)

HQL aggregate functions

- **Hibernate supports multiple aggregate functions. when they are used in HQL queries, they return an aggregate value (such as sum, average, and count) calculated from property values of all objects satisfying other query criteria.**
- **These functions can be used along with the distinct and all options, to return aggregate values calculated from only distinct values and all values (except null values), respectively.**

HQL aggregate functions

- **Following is a list of aggregate functions with their respective syntax; all of them are self-explanatory.**

  count([distinct | all] object | object.property) count(*) (equivalent to count(all ...), counts null values also)

  sum ([distinct | all] object.property)

  avg([distinct | all] object.property)

  max([distinct | all] object.property)

  min([distinct | all] object.property)

HQL aggregate functions

- **Example:**

  “select cat.weight + sum(kitten.weight) from Cat cat join cat.kittens kitten group by cat.id, cat.weight “

  select distinct cat.name from Cat cat

  select count(distinct product.supplier.name) from Product product

Using Restrictions with HQL

- **As with SQL, you use the where clause to select results that match your query's expressions. HQL provides many different expressions that you can use to construct a query. In the HQL language grammar, there are the following possible expressions:**
  - Logic operators: OR, AND, NOT
  - Equality operators: =, <>, !=, ^=
  - Comparison operators: <, >, <=, >=, like, not like, between, not between

Using Restrictions with HQL

- Math operators: +, -, *, /
- Concatenation operator: ||
- Cases: Case when <logical expression> then <unary expression> else _<unaryexpression> end
- Collection expressions: some, exists, all, any

➢ **Example:**

- from Product where price > 25.0 and name like 'Mou%‘
- from DomesticCat cat where cat.name between 'A' and 'B'
- from DomesticCat cat where cat.name in ('Foo', 'Bar', 'Baz')

Using Named Parameters

- **Hibernate supports named parameters in its HQL queries. This makes writing queries that accept input from the user easy—and you do not have to defend against SQL injection attacks.**
- **SQL injection is an attack against applications that create SQL directly from user input with string concatenation. For instance, if we accept a name from the user through a web application form, then it would be very bad form to construct an SQL (or HQL) query like this:**
  - String sql = "select p from products where name = '" + name + "'";

Using Named Parameters

- **Hibernate's named parameters are similar to the JDBC query parameters (?) you may already be familiar with, but Hibernate's parameters are less confusing. It is also more straightforward to use Hibernate's named parameters if you have a query that uses the same parameter in multiple places**
- **The simplest example of named parameters uses regular SQL types for the parameters:**

```
String hql = "from Product where price > :price";
Query query = session.createQuery(hql);
query.setDouble("price",25.0);
List results = query.list();
```

Using Named Parameters

- **When the value to be provided will be known only at run time, you can use some of HQL's object-oriented features to provide objects as values for named parameters. The Query interface has a setEntity() method that takes the name of a parameter and an object.**

```
String supplierHQL = "from Supplier where name='MegaInc'";
Query supplierQuery = session.createQuery(supplierHQL);
Supplier supplier = (Supplier) supplierQuery.list().get(0);
String hql = "from Product as product where product.supplier=:supplier";
Query query = session.createQuery(hql);
query.setEntity("supplier",supplier);
List results = query.list()
```

Named HQL Query in Mapping File

```
<hibernate-mapping>
   <class name="Product">
      <id name="id" type="int">
         <generator class="increment"/>
      </id>
      <property name="name" type="string"/>
      <property name="description" type="string"/>
      <property name="price" type="double"/>
<many-to-one name="supplier" class="Supplier" column="supplierId"/>
   </class>
   <query name="HQLpricing"><![CDATA[
select product.price from Product product where product.price > 25.0]]>
         </query>
   <sql-query name="SQLpricing">
      <return-scalar column="price" type="double"/>
      <![CDATA[
select product.price from Product as product where product.price > 25.0]]>
   </sql-query>
</hibernate-mapping>
```

Get Named HQL Query from Mapping File

```
Session session = HibernateUtil.currentSession();
   Query query = session.getNamedQuery("HQLpricing");
      List results = query.list();
      displayObjectList(results);
static public void displayObjectList(List list)
   {
      Iterator iter = list.iterator();
      if (!iter.hasNext())
      {
         System.out.println("No objects to display.");
         return;
      }
      while (iter.hasNext())
      {
         System.out.println("New object");
         Object obj = (Object) iter.next();
         System.out.println(obj); }}
```

Obtaining a Unique Result

- **HQL's Query interface provides a uniqueResult() method for obtaining just one object from an HQL query.**

```
Query query = session.createQuery(hql);
query.setMaxResults(1);
Product product = (Product) query.uniqueResult();
//test for null here if needed
```

Sorting Results with the order by Clause

- **To sort your HQL query's results, you will need to use the order by clause. You can order the results by any property on the objects in the result set: either ascending (asc) or descending (desc). You can use ordering on more than one property in the query if you need to. A typical HQL query for sorting results looks like this:**

  from Product p where p.price>25.0 order by p.price desc

  If you wanted to sort by more than one property, you would just add the additional properties to the end of the order by clause, separated by commas. For instance, you could sort by product price and the supplier's name, as follows:

  from Product p order by p.supplier.name asc, p.price asc

Using Multiple Table

- **Associations allow you to use more than one class in an HQL query, just as SQL allows you to use joins between tables in a relational database. Add an association to an HQL query with the join clause.**
- **Hibernate supports five different types of joins: inner join, cross join, left outer join, right outer join, and full outer join.**
- **If you use cross join, just specify both classes in the from clause (from Product p, Supplier s).**
- **For the other joins, use a join clause after the from clause. Specify the type of join, the object property to join on, and an alias for the other class.**

Using Multiple Table

- **Example1:**
  “select s.name, p.name, p.price from Product p inner join p.supplier as s”

- **Example2:**
  “from Cat as cat inner join cat.mate as mate left outer join cat.kittens as kitten “

- **You may supply extra join conditions using the HQL with keyword.**

- **Example 3:**
  “from Cat as cat left join cat.kittens as kitten with kitten.bodyWeight > 10.0 “

Demo

- **HibernateHQLQuery**
- **HibernateHQLNamedHQLQuery**

HQL Performance: Lazy Loading

- **If you would like to start optimizing performance, you can ask Hibernate to fetch the associated objects and collections for an object in one query.**
- **A "fetch" join allows associations or collections of values to be initialized along with their parent objects using a single select. This is particularly useful in the case of a collection. It effectively overrides the outer join and lazy declarations of the mapping file for associations and collections**
- **If you were using lazy loading with Hibernate, the objects in the collection would not be initialized until you accessed them. If you use fetch on a join in your query, you can ask Hibernate to retrieve the objects in the collection at the time the query executes.**

"from Supplier s inner join fetch s.products as p"

HQL Performance: Bulk Updates and Deletes with HQL

- **The Query interface now contains a method called executeUpdate() for executing HQL UPDATE or DELETE statements. The executeUpdate() method returns an int that contains the number of rows affected by the update or delete, as follows:**

public int executeUpdate() throws HibernateException

Bulk Updates and Deletes with HQL

```
String hql = "update Supplier set name = :newName
 where name = :name";

    Query query = session.createQuery(hql);
    query.setString("name","SuperCorp");
    query.setString("newName","MegaCorp");
    int rowCount = query.executeUpdate();
    System.out.println("Rows affected: " + rowCount);

//See the results of the update
    query = session.createQuery("from Supplier");
    List results = query.list();
```

Query by criteria

- **The Criteria Query API lets you build nested, structured query expressions in Java, providing a compile-time syntax-checking that is not possible with a query language like HQL or SQL.**
- **The Criteria API also includes query by example (QBE) functionality—this lets you supply example objects that**

  contain the properties you would like to retrieve instead of having to spell the components of the query out step by step.
- **It also includes projection and aggregation methods, including counts.**

Query by criteria

- **The Hibernate Session interface contains several createCriteria() methods.**
- **Pass the persistent object's class or its entity name to the createCriteria() method, and Hibernate will create a Criteria object that returns instances of the persistence object's class when your application executes a criteria query.**
- **The simplest example of a criteria query is one with no optional parameters or restrictions—the criteria query will simply return every object that corresponds to the class.**

  "Criteria crit = session.createCriteria(Product.class);

  List results = crit.list();"

Using Restrictions with Criteria

- **The Criteria API makes it easy to use restrictions in your queries to selectively retrieve objects.**
- **You may add these restrictions to a Criteria object with the add() method. The add() method takes an org.hibernate.criterion.Criterion object that represents an individual restriction. You can have more than one restriction for a criteria query.**

"Criteria crit = session.createCriteria(Product.class);

crit.add(Restrictions.eq("name","Mouse"));

List results = crit.list()"

Using like() or ilike()

- ***Criteria crit = session.createCriteria(Product.class);***

  crit.add(Restrictions.like("name","Mou%"));

  List results = crit.list();
- **org.hibernate.criterion.MatchMode is used specify how to match the specified vlaue to the stored data.The MatchMode Object has four different matches:**
  - **ANYWHERE:** Anyplace in the string
  - **END:** The end of the string
  - **EXACT:** An exact match
  - **START:** The beginning of the string

Using like() or ilike()

- **Example that uses the ilike() method to search for case-insensitive matches at the end of the string**

"Criteria crit = session.createCriteria(Product.class);

crit.add(Restrictions.ilike("name","browser", MatchMode.END));

List results = crit.list();"

Sample Code using gt(),isNull()

- **The isNull() and isNotNull() restrictions allow you to do a search for objects that have (or do not have) null property values. This is easy to demonstrate.**

```
Criteria crit = session.createCriteria(Product.class);
 crit.add(Restrictions.isNull("name"));
 List results = crit.list();
"crit.add(Restrictions.gt("price",new Double(25.0)));
List results = crit.list();"
```

Sample Code using gt(),isNull()

- **When you add more than one constraint to a criteria query, it is interpreted as an AND, like so**

"Criteria crit = session.createCriteria(Product.class);

crit.add(Restrictions.gt("price",new Double(25.0)));

crit.add(Restrictions.like("name","K%"));

List results = crit.list();"

Restrictions.or()

- ***If we wanted to create an OR expression with more than two different criteria (for example, price >25.0 OR name like Mou%***

  OR description not like blocks%), we would use an org.hibernate.

  criterion.

```
Criterion price = Restrictions.gt("price",new Double(25.0));
Criterion name = Restrictions.like("name","Mou%");
LogicalExpression orExp = Restrictions.or(price,name);
crit.add(orExp);
List results = crit.list();
```

Using Restrictions.disjunction();

- ***Disjunction object to represent a disjunction.*** **The disjunction is more convenient than building a tree of OR expressions in code.**

"Criteria crit = session.createCriteria(Product.class);

Criterion price =Restrictions.gt("price", new Double(25.0));

Criterion name = Restrictions.like("name","Mou%");

Criterion desc = Restrictions.ilike("description","blocks%");

Disjunction disjunction = Restrictions.disjunction();

disjunction.add(price);

disjunction.add(name);

disjunction.add(desc);

crit.add(disjunction);

List results = crit.list();"

Paging Through the Result Set

- **One common application pattern that criteria can address is pagination through the result set of a database query. When we say pagination, we mean an interface in which the user sees part of the result set at a time, with navigation to go forward and backward through the results.**

  "Criteria crit = session.createCriteria(Product.class);

  crit.setFirstResult(1);

  crit.setMaxResults(2);

  List results = crit.list();"

- **Obtaining a Unique Result**

  "Criteria crit = session.createCriteria(Product.class);

  Criterion price = Restrictions.gt("price",new Double(25.0));

  crit.setMaxResults(1);

  Product product = (Product) crit.uniqueResult();"

Sorting the Query's Results

- **The Criteria API provides the org.hibernate.criterion.Order class to sort your result set in either ascending or descending order, according to one of your object's properties.**

"Criteria crit = session.createCriteria(Product.class);

crit.add(Restrictions.gt("price",new Double(25.0)));

crit.addOrder(Order.desc("price"));

List results = crit.list();"

Hibernate Projections

- **Instead of working with objects from the result set, you can treat the results from the result set as a set of rows and columns, also known as a *projection* of the data.**

```
Criteria crit = session.createCriteria(Product.class);
crit.setProjection(Projections.rowCount());
List results = crit.list();
```

- **The results list will contain one object, an Integer that contains the results of executing the COUNT SQL statement.**

Hibernate Projections

- **Other aggregate functions available through the Projections factory class include the following:**
- **avg(String propertyName): Gives the average of a property's value**
- **count(String propertyName): Counts the number of times a property occurs**
- **countDistinct(String propertyName): Counts the number of unique values the - property contains**
- **max(String propertyName): Calculates the maximum value of the property values**
- **min(String propertyName): Calculates the minimum value of the property values**
- **sum(String propertyName): Calculates the sum total of the property values**

Hibernate Projections

```
Criteria crit = session.createCriteria(Product.class);
ProjectionList projList = Projections.projectionList();
projList.add(Projections.max("price"));
projList.add(Projections.min("price"));
projList.add(Projections.avg("price"));
projList.add(Projections.countDistinct("description"));
crit.setProjection(projList);
List results = crit.list();
```

SQL's GROUP BY clause

- **You can group your results with the groupProperty projection**

```
Criteria crit = session.createCriteria(Product.class);
ProjectionList projList = Projections.projectionList();
projList.add(Projections.groupProperty("name"));
projList.add(Projections.groupProperty("price"));
crit.setProjection(projList);
List results = crit.list();
```

Review

- **In this lesson, we learned**
  - Hibernate Query Language(HQL)
  - Hibernate Query Language(HQL) Performance
    - Lazy Loading
    - Bulk updates
  - Query by criteria
  - Hibernate Projections