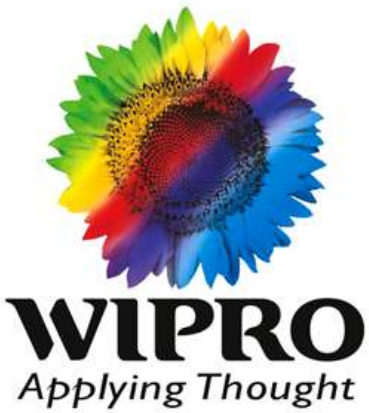


Hibernate- V

Querying

Agenda

1. Querying
2. CRUD Operations with Hibernate

Querying

Objectives

At the end of this module, you will be able to

- Query with Hibernate Query Language (HQL)
- Use Criteria Queries
- Create queries with Native SQL

Querying in Hibernate

Hibernate provides with 3 different ways of querying the database

- Hibernate Query Language (HQL)
- Criteria Queries
- Native SQL

Hibernate Query Language (HQL)

- HQL is a powerful query language
  - Used to execute queries against database
- HQL is much like SQL and is case-insensitive, except for the 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
- Hibernate Query Language uses classes and properties instead of tables and columns

HQL (Contd.).

- *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)

Hibernate Query Language (HQL)

Features of HQL

- Full support for relational operations**:** HQL allows representing SQL queries in the form of objects. It 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 eliminates the need of creating the object and populating the data from result set
- Polymorphic Queries**:** HQL fully supports **polymorphic queries**. Polymorphic queries gives 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

HQL (Contd.).

Understanding HQL Syntax

- Any Hibernate Query Language may consist of following elements:
  - Clauses
  - Aggregate functions
  - Subqueries
- Clauses in the HQL are:
  - from
  - select
  - where
  - order by
  - group by

HQL (Contd.).

- Aggregate functions are:
  - avg(...), sum(...), min(...), max(...)
  - count(*)
  - count(...), count(distinct ...), count(all...)

- Subqueries
  Subqueries are nothing but a query within another query.
  Hibernate supports Subqueries if the underlying database supports it

HQL (Contd.).

Parameter Binding

HQL allows you to bind some parameters in the query, if you need to give different set of values to the same query.

HQL supports 2 types of parameter binding namely:

1. Named Parameter Binding

```
Query myQuery = hibernateSession. createQuery ("from
                                Employee as emp where emp.firstName = :name");
myQuery.setString("name", “John");
Iterator employees = myQuery.iterate();
```

2. JDBC Parameter Binding

```
Query myQuery = hibernateSession. createQuery("from
                                 Student as emp where emp.firstName = ?");
myQuery.setString(0, “John");
Iterator employees = myQuery.iterate();
```

HQL (Contd.).

- Paging through the query results

```
Query query = session.createQuery("from Chapter");
query.setFirstResult(4);   // Starting row
query.setMaxResults(3);    // Size of each page
ChapterList = (List<Chapter>) query.list();
```

- Perform HQL Query with like

```
String hql = "from Chapter where title like '%s'";
Query query = session.createQuery(hql);
ChapterList = (List<Chapter>) query.list();
```

Criteria Queries

With Criteria Query API you can 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

```
Criteria crit = session.createCriteria(Chapter.class);
crit.add(Restrictions.like("title", "%s"));
ChapterList = (List<Chapter>) crit.list();
```

Restrictions are also useful for doing math comparisons:

- greater-than comparison is `gt()`
- greater-than-or-equal-to comparison is `ge()`
- less-than comparison is `lt()`
- less-than-or-equal-to comparison is `le()`

Native SQL

When there are some features that are specific to a particular database and you want to use these features in your application, you should chose Native SQL

You can create a native SQL query from the session with the createSQLQuery() method on the Session interface

```
String sql = "select {chapter.*} from Chapter chapter";
SQLQuery query = session.createSQLQuery(sql);
query.addEntity("chapter", Chapter.class);
ChapterList = query.list();
```

Quiz

- Which of the following is not a way of querying the database with Hibernate?

  a. Hibernate Query Language (HQL)

  b. Criteria Queries

  c. Native SQL

  d. HSQL

  Ans: d

Summary

At the end of this module, you were able to

- Query with Hibernate Query Language (HQL)
- Use Criteria Queries
- Create queries with Native SQL

CRUD Operations with annotations

Objectives

At the end of this module, you will be able to

- Develop an application containing CRUD operations, using annotations

Demo Application using annotations

Now we will see, how to create a java application that performs all the CRUD operations using annotations provided by hibernate framework.

In the previous CRUD operations example, we used an xml file for entity mappings. In this case, we will use annotations instead of this xml file.

Here we will need only one xml file, i.e. hibernate.cfg.xml, which will contain the usual configuration information.

All the mappings related to entities, like table definition, column mappings etc. are defined within the entity class, using annotations.

Demo Application using annotations

We will develop an application called Inventory Management, which will have options for inserting, updating deleting and retrieving item details.

We will define three java classes and an xml file hibernate.cfg.xml.

The three java files are :

1) Item.java – The entity class
2) DataOperations.java – Which will have methods for performing CRUD operations
3) ItemClient.java – Which will have the main method containing a menu for invoking these methods.

Demo Application using annotations

Item.java

```
package simple.hibernate;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="NewItem")
public class Item {
	@Id
	@Column(name="itemcode")
	private int itemno;

	@Column(name="item_name", length=15)
	private String itemname;
```

Contd..

Demo Application using annotations (Contd.).

```
private int quantity;
private Date procuredate;

@Column(columnDefinition="Decimal(10,2)")
private float unitprice;

public Date getProcuredate() {
        return procuredate;
}
public void setProcuredate(Date procuredate) {
        this.procuredate = procuredate;
}

public int getItemno() {
        return itemno;
}
public void setItemno(int itemno) {
        this.itemno = itemno;
}
```

Contd…

Demo Application using annotations (Contd.).

```
        public String getItemname() {
                return itemname;
        }
        public void setItemname(String itemname) {
                this.itemname = itemname;
        }

        public int getQuantity() {
                return quantity;
        }
        public void setQuantity(int quantity) {
                this.quantity = quantity;
        }
        public float getUnitprice() {
                return unitprice;
        }
        public void setUnitprice(float unitprice) {
                this.unitprice = unitprice;
        }
}
```

Contd..

Demo Application using annotations (Contd.).

DataOperations.java

```
package simple.hibernate;

import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

import java.sql.Date;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import simple.hibernate.Item;

public class DataOperations {
        public void insert(){
                Scanner x = new Scanner(System.in);
                System.out.println("Enter Item Number : ");
                int itemno = x.nextInt();
```

Contd..

Demo Application using annotations (Contd.).

```
        System.out.println("Enter Item Name : ");
        String itemname = x.next();
        System.out.println("Enter Quantity : ");
        int quantity = x.nextInt();
        System.out.println("Enter Item Price : ");
        float unitprice = x.nextFloat();
        System.out.println("Enter Procurement Date (yyyy-mm-dd format only) : ");
        String sdate = x.next();
        Date procuredate = Date.valueOf(sdate);
        Item i=new Item();
        i.setItemno(itemno);
        i.setItemname(itemname);
        i.setQuantity(quantity);
        i.setUnitprice(unitprice);
        i.setProcuredate(procuredate);
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session sess=sf.openSession();
        sess.beginTransaction();
        sess.save(i);
        sess.getTransaction().commit();
}
```

Contd..

Demo Application using annotations (Contd.).

```
public int update(){
        Scanner x = new Scanner(System.in);
        System.out.println("Enter Item Number : ");
        int itemno = x.nextInt();

        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();

        session.beginTransaction();
        Item i = (Item)session.get(Item.class, itemno);

        System.out.println("The item no is : "+i.getItemno());
        System.out.println("The item name is : "+i.getItemname());
        System.out.println("The Quantity is : "+i.getQuantity());
        System.out.println("The Unit Price is : "+i.getUnitprice());
        System.out.println("The Procurement Date is : "+i.getProcuredate());

        System.out.println("Enter Quantity : ");
        int quantity = x.nextInt();
        System.out.println("Enter Item Price : ");
        float unitprice = x.nextFloat();
```

Contd..

Demo Application using annotations (Contd.).

```
        i.setItemno(itemno);
        i.setItemname(i.getItemname());
        i.setQuantity(quantity);
        i.setUnitprice(unitprice);
        i.setProcuredate(i.getProcuredate());
        session.saveOrUpdate(i);
        session.getTransaction().commit();
        return 0;
}
public int delete(){
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();
        Scanner x = new Scanner(System.in);
        System.out.println("Enter Item Number : ");
        int itemno = x.nextInt();
        Item i=new Item();
        i.setItemno(itemno);
        session.beginTransaction();
        session.delete(i);
        session.getTransaction().commit();
        return 0;
}
```

Contd..

Demo Application using annotations (Contd.).

```
public List<Item> selectOne() {
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();

        Scanner x = new Scanner(System.in);
        System.out.println("Enter Item Number : ");
        int itemno = x.nextInt();

        Item item = (Item)session.get(Item.class, itemno);
        System.out.println("Item No :"+item.getItemno());
        System.out.println("Item Name :"+item.getItemname());
        System.out.println("Quantity :"+item.getQuantity());
        System.out.println("Unit Price :"+item.getUnitprice());
        System.out.println("Procurement Date :"+item.getProcuredate());

        return null;
}
```

Contd..

Demo Application using annotations (Contd.).

```
        public List<Item> selectAll() {
            SessionFactory sf=new Configuration().configure().buildSessionFactory();
            Session session=sf.openSession();

            List<Item> l = (List<Item>)session.createCriteria(Item.class).list();

            for (Iterator iterator = l.iterator(); iterator.hasNext();) {
                Item item = (Item) iterator.next();
                System.out.println("Item No :"+item.getItemno());
                System.out.println("Item Name :"+item.getItemname());
                System.out.println("Quantity :"+item.getQuantity());
                System.out.println("Unit Price :"+item.getUnitprice());
                System.out.println("Procurement Date :"+item.getProcuredate());
            }
            return l;
        }
}
```

Demo Application using annotations (Contd.).

ItemClient.java

```
package simple.hibernate;
import java.util.Scanner;

public class ItemClient {
        public static void main(String[] args) {
                boolean b = true;
                int i=0;
                Scanner x = null;
                DataOperations d;
                do {
                        System.out.println("1. Add Record");
                        System.out.println("2. Modify Record");
                        System.out.println("3. Delete Records");
                        System.out.println("4. Display One Record");
                        System.out.println("5. Display All Records");
                        System.out.println("Specify what you want to do..!");
                        x = new Scanner(System.in);
                        d = new DataOperations();
                        i = x.nextInt();
```

Contd..

Demo Application using annotations (Contd.).

```
                switch(i)
                {
                case 1:d.insert();
                break;
                case 2:d.update();
                break;
                case 3:d.delete();
                break;
                case 4:d.selectOne();
                break;
                case 5:d.selectAll();
                break;
                case 6:b=false;
                break;
                default:System.out.println("Invalid Selection : Enter 1 or 2 or 3 or 4 or 5 or 6 for
                exit");
                b=false;
                }
            }while(b);
            System.out.println("Exiting Application");
        }
}
```

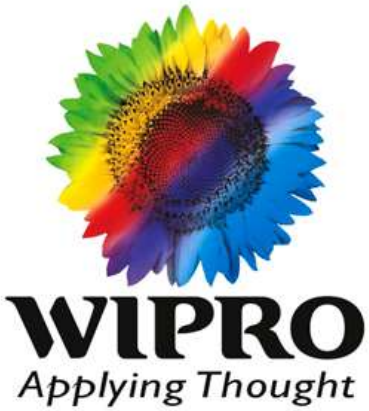
Demo Application using annotations (Contd.).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
     <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
     <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger1</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping class="simple.hibernate.Item"/>
   </session-factory>
</hibernate-configuration>
```

Summary

At the end of this module, you were able to

- Develop an application containing CRUD operations, using annotations



Thank You