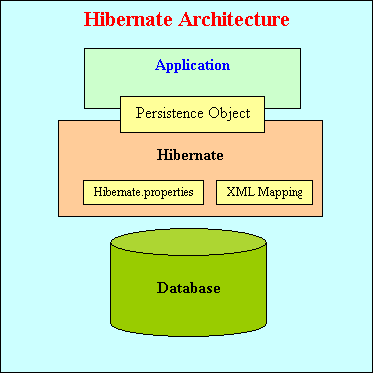
**What is Hibernate?**

Hibernate maps the Java classes to the database tables. It also provides the data query and retrieval facilities that significantly reduces the development time.  Hibernate is not the best solutions for data centric applications that only uses the stored-procedures to implement the business logic in database. It is most useful with object-oriented domain modes and business logic in the Java-based middle-tier. Hibernate allows transparent persistence that enables the applications to switch any database. Hibernate can be used in Java Swing applications, Java Servlet-based applications, or J2EE applications using EJB session beans.

**Features of Hibernate**

* Hibernate 3.0 provides three full-featured query facilities: **Hibernate Query Language**, the newly enhanced **Hibernate Criteria Query API**, and enhanced support for queries expressed in the **native SQL**dialect of the database.
* Enhanced Criteria query API: with full support for projection/aggregation and subselects.
* Eclipse support, including a suite of Eclipse plug-ins for working with Hibernate 3.0, including mapping editor, interactive query prototyping, schema reverse engineering tool.
* Hibernate is Free under LGPL: Hibernate can be used to develop/package and distribute the applications for free.
* Hibernate is Scalable: Hibernate is very performant and due to its dual-layer architecture can be used in the clustered environments.
* Less Development Time: Hibernate reduces the development timings as it supports inheritance, polymorphism, composition and the Java Collection framework.
* Automatic Key Generation: Hibernate supports the automatic generation of primary key for your tables.
* EJB3-style persistence operations: EJB3 defines the create() and merge() operations, which are slightly different to Hibernate's saveOrUpdate() and saveOrUpdateCopy()operations. Hibernate3 will support all four operations as methods of the Session interface.
* Hibernate XML binding enables data to be represented as XML and POJOs interchangeably.
* The EJB3 draft specification support for POJO persistence and annotations.

**Hibernate Architecture**



The above diagram shows that Hibernate is using the database and configuration data to provide persistence services (and persistent objects) to the application.

To use Hibernate, it is required to create Java classes that represents the table in the database and then map the instance variable in the class with the columns in the database. Then Hibernate can be used to perform operations on the database like select, insert, update and delete the records in the table. Hibernate automatically creates the query to perform these operations.

Hibernate architecture has three main components:

**Connection Management**  
Hibernate Connection management service provide efficient management of the database connections. Database connection is the most expensive part of interacting with the database as it requires a lot of resources of open and close the database connection.

**Transaction management:**  
Transaction management service provide the ability to the user to execute more than one database statements at a time.

**Object relational mapping:**  
Object relational mapping is technique of mapping the data representation from an object model to a relational data model. This part of the hibernate is used to select, insert, update and delete the records form the underlying table. When we pass an object to a **Session.save()** method, Hibernate reads the state of the variables of that object and executes the necessary query.

Hibernate is very good tool as far as object relational mapping is concern, but in terms of connection management and transaction management, it is lacking in performance and capabilities. So usually hibernate is being used with other connection management and transaction management tools. For example apache DBCP is used for connection pooling with the Hibernate.

Hibernate provides a lot of flexibility in use. It is called "**Lite**" architecture when we only uses the object relational mapping component. While in "**Full Cream**" architecture all the three component Object Relational mapping, Connection Management and Transaction Management) are used.

**Configuring Hibernate**

Hibernate uses the hibernate.cfg.xml to create the connection pool and setup required environment.

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
"-//Hibernate/Hibernate Configuration DTD//EN"  
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
  
<hibernate-configuration>  
<session-factory>  
  <property name="hibernate.connection.driver\_class">

com.mysql.jdbc.Driver</property>  
  <property name="hibernate.connection.url">

**jdbc:mysql://localhost/hibernatetutorial**</property>  
  <property name="hibernate.connection.username">**root**</property>  
  <property name="hibernate.connection.password"></property>  
  <property name="hibernate.connection.pool\_size">10</property>  
  <property name="show\_sql">**true**</property>  
  <property name="dialect">**org.hibernate.dialect.MySQLDialect**</property>  
  <property name="hibernate.hbm2ddl.auto">**update**</property>  
  <!-- Mapping files -->  
  <mapping resource="**contact.hbm.xml**"/>  
</session-factory>  
</hibernate-configuration>

In the above configuration file we specified to use the "**hibernatetutorial"** which is running on **localhost**and the user of the database is root with no password. The **dialect** property  is**org.hibernate.dialect.MySQLDialect** which tells the Hibernate that we are using MySQL Database. Hibernate supports many database. With the use of the [**Hibernate**](http://www.hibernate.org/) (Object/Relational Mapping and Transparent Object Persistence for Java and SQL Databases),  we can use the following databases dialect type property:

* DB2 - org.hibernate.dialect.DB2Dialect
* HypersonicSQL - org.hibernate.dialect.HSQLDialect
* Informix - org.hibernate.dialect.InformixDialect
* Ingres - org.hibernate.dialect.IngresDialect
* Interbase - org.hibernate.dialect.InterbaseDialect
* Pointbase - org.hibernate.dialect.PointbaseDialect
* PostgreSQL - org.hibernate.dialect.PostgreSQLDialect
* Mckoi SQL - org.hibernate.dialect.MckoiDialect
* Microsoft SQL Server - org.hibernate.dialect.SQLServerDialect
* MySQL - org.hibernate.dialect.MySQLDialect
* Oracle (any version) - org.hibernate.dialect.OracleDialect
* Oracle 9 - org.hibernate.dialect.Oracle9Dialect
* Progress - org.hibernate.dialect.ProgressDialect
* FrontBase - org.hibernate.dialect.FrontbaseDialect
* SAP DB - org.hibernate.dialect.SAPDBDialect
* Sybase - org.hibernate.dialect.SybaseDialect
* Sybase Anywhere - org.hibernate.dialect.SybaseAnywhereDialect

The <mapping resource="**contact.hbm.xml**"/> property is the mapping for our contact table.

**Writing First Persistence Class**  
Hibernate uses the Plain Old Java Objects (POJOs) classes to map to the database table. We can configure the variables to map to the database column. Here is the code for Contact.java:

**Mapping the Contact Object to the Database Contact table**

This can easily by done using annotations. Read them for more details.

Hibernate Session is the main runtime interface between a Java application and Hibernate. First we are required to get the Hibernate Session.SessionFactory allows application to create the Hibernate Sesssion by reading the configuration from hibernate.cfg.xml file.  Then the save method on session object is used to save the contact information to the database:

**session.save(contact)**

# Understanding Hibernate <generator> element

In this lesson you will learn about hibernate <generator> method in detail. Hibernate generator element generates the primary key for new record. There are many options provided by the generator method to be used in different situations.

Here are the list of some commonly used generators in hibernate:

(Study about various Generators separately)………

# Hibernate Update Query

Retrieve the row using session.get(Class Type, ID) and then set values in it…. After that use session.update(Object)…and commit the transaction.

# Hibernate Delete Query

Retrieve the row using session.get(Class Type, ID) …. After that use session.delete(Object)…and commit the transaction.

# Hibernate Query Language

# Hibernate Query Language or HQL for short is extremely powerful query language. HQL is much like SQL  and are case-insensitive, except for the names of the Java Classes and properties.

# 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.

# There are other options that can be used while using Hibernate. These are Query By Criteria (QBC)and Query BY Example (QBE) using Criteria API and the Native SQL 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: Understand by studying….

# 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).

**Understanding HQL Syntax:**

Any Hibernate Query Language may consist of following elements:

* Clauses
* Aggregate functions
* Subqueries

**Clauses in the HQL are:**

* From: using from clause, we get the values. Example: session.createQuery("From Student").list();

This will return all values in table corresponding to Student POJO.

* Select:  The select clause picks up objects and properties to return in the query result set.  Example: “Select s.firstName from Student s”.

This will return firstName from table corresponding to Student POJO.

* Where: The where clause specifies the condition depending on which, result set is to be generated. Example: From Student where firstName = ‘Manish’”.
* Order by : Order by clause is used to retrieve the data from database in the sorted order by any property of returned class or components. Example: “From Student Order by firstName”.
* Group by: Group by clause is used to return the aggregate values by grouping on returned component. Example: Select avg(marks) from Subjects Group by Subject ID.

Aggregate Functions are:

* Avg(); Syntax: avg([distinct | All] property.)

Example: select avg(investementAmount) from Insurance insurance

Here property is property whose average is to be calculated.

* Same for min,max, sum.
* Count (\*) returns number of rows satisfying this condition.

# Hibernate Criteria Query Example

The Criteria interface allows to create and execute object-oriented queries. It is powerful alternative to the HQL but has own limitations. Criteria Query is used mostly in case of multi criteria search screens, where HQL is not very effective.

The interface org.hibernate.Criteria is used to create the criterion for the search. The org.hibernate.Criteria interface represents a query against a persistent class. The Session is a factory for Criteria instances.

Different method provided by Criteria interface can be used with the help of Restrictions to restrict the records fetched from database.

**Criteria Interface provides the following methods:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| add | The Add method adds a Criterion to constrain the results to be retrieved. |
| addOrder | Add an Order to the result set. |
| createAlias | Join an association, assigning an alias to the joined entity |
| createCriteria | This method is used to create a new Criteria, "rooted" at the associated entity. |
| setFetchSize | This method is used to set a fetch size for the underlying JDBC query. |
| setFirstResult | This method is used to set the first result to be retrieved. |
| setMaxResults | This method is used to set a limit upon the number of objects to be retrieved. |
| uniqueResult | This method is used to instruct the Hibernate to fetch and return the unique records from database. |

Class Restriction provides built-in criterion via static factory methods. Important methods of the Restriction class are:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Restriction.**allEq** | This is used to apply an "equals" constraint to each property in the key set of a Map |
| Restriction.**between** | This is used to apply a "between" constraint to the named property |
| Restriction.**eq** | This is used to apply an "equal" constraint to the named property |
| Restriction.**ge** | This is used to apply a "greater than or equal" constraint to the named property |
| Restriction.**gt** | This is used to apply a "greater than" constraint to the named property |
| Restriction.**idEq** | This is used to apply an "equal" constraint to the identifier property |
| Restriction.**ilike** | This is case-insensitive "like", similar to Postgres ilike operator |
| Restriction.**in** | This is used to apply an "in" constraint to the named property |
| Restriction.**isNotNull** | This is used to apply an "is not null" constraint to the named property |
| Restriction.**isNull** | This is used to apply an "is null" constraint to the named property |
| Restriction.**le** | This is used to apply a "less than or equal" constraint to the named property |
| Restriction.**like** | This is used to apply a "like" constraint to the named property |
| Restriction.**lt** | This is used to apply a "less than" constraint to the named property |
| Restriction.**ltProperty** | This is used to apply a "less than" constraint to two properties |
| Restriction.**ne** | This is used to apply a "not equal" constraint to the named property |
| Restriction.**neProperty** | This is used to apply a "not equal" constraint to two properties |
| Restriction.**not** | This returns the negation of an expression |
| Restriction.**or** | This returns the disjuction of two expressions |

# Hibernate Projections

**Projections**: The package **Criteria** is used as a framework by the applications just to build the new kinds of projection. may be used by applications as a framework for building new kinds of **Projection.** In general Projection means to retrieve while in case of SQL Projection means "**Select**" clause. Most of the applications uses the built-in projection types by means of the static factory methods of this class.

NOTE: This is basically like creating view of a table.

To get Unique results, use criteria.uniqueResults() instead of using criteria.list().

## Hibernate Criteria Join API

Hibernate Criteria JOIN API allows users to perform join operation.

Suppose you have to perform a operation like

SELECT S.\*, C.\* FROM STUDENT S, CONTACT  C WHERE S.ROLL\_NO=C.ID;

Then you can write this statement using Criteria in a very simple way

Criteria criteria = session.createCriteria(Student.class);  
criteria.setFetchMode("Contact", FetchMode.JOIN);  
List list = criteria.list();

The above statement is very simple, that you have to only call the setFetchMode(ClassName, Type of operation);

In the setFetchMode() method first provide the name of the class, and type of operation on the class.

Suppose you have to apply restriction on join operation such as

SELECT C.\*, S.\* FROM CONTACT C, STUDENT S WHERE C.ID=S.ROLL\_NO AND S.ROLL\_NO=?5?;

Then you can write the above statement in criteria as

Criteria criteria = session.createCriteria(Student.class);  
        criteria.setFetchMode("Contact", FetchMode.JOIN).add(  
                        Restrictions.eq("id", 2));  
        List list = criteria.list();

The eq() method of restriction class compares the result with id and returns.