**Abstract:** Hibernate is popular open source object relational mapping tool for Java platform. It provides powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework.  
  
Hibernate takes care of the mapping from Java classes to database tables and also provides data query and retrieval facilities. Hibernate significantly reduces development time in the software projects.

**Topics Covered in the Hibernate Tutorial:**

* [Introduction to Hibernate 3.0](http://www.roseindia.net/hibernate/introduction.shtml)  
  This lesson introduces you with the Hibernate 3.0 and provides the detailed features of the Hibernate 3.0
* [Hibernate Architecture](http://www.roseindia.net/hibernate/hibernate_architecture.shtml)  
  In this lesson you will learn the architecture of Hibernate.
* [First Hibernate Application](http://www.roseindia.net/hibernate/firstexample.shtml)  
  This section describes you how to develop sample Hibernate Application.
* [Running the Example in Eclipse](http://www.roseindia.net/hibernate/runningexample.shtml)  
  This section shows you how to run the example in the Eclipse.
* [Understanding Hibernate O/R Mapping](http://www.roseindia.net/hibernate/hibernateormapping.shtml)  
  This section describes the each component of the hibernate mapping file..
* [Understanding Hibernate <generator> element](http://www.roseindia.net/hibernate/hibernateidgeneratorelement.shtml)  
  In this lesson you will learn about hibernate <generator> method in detail. Hibernate generator element generates the primary key for new record.
* [Using Hibernate <generator> to generate id incrementally](http://www.roseindia.net/hibernate/incrementgenerator.shtml)  
  In this lesson I will show you how to write running program to demonstrate it.  
      
     
  **Hibernate Query Language**
* [Introduction to Hibernate Query Language](http://www.roseindia.net/hibernate/hibernatequerylanguagehql.shtml)  
  In this lesson you will learn about Hibernate Query Language and its features.
* [Preparing table for HQL Examples](http://www.roseindia.net/hibernate/preparinginsurancedatabase.shtml)  
  In this lesson you will create insurance table and populate it with the data for future examples.
* [Developing POJO class](http://www.roseindia.net/hibernate/writingobjectrelationalmappings.shtml)  
  In this lesson we will write the java class and add necessary code in the contact.hbm.xml file.
* [HQL from Clause Example](http://www.roseindia.net/hibernate/hibernatefromclauseexample.shtml)  
  The from clause is the simplest possible Hibernate Query. In this example you will learn how to use the HQL from clause.
* [HQL Select Clause Example](http://www.roseindia.net/hibernate/selectclause.shtml)  
  In this lesson we will write example code to select the data from Insurance table using Hibernate Select Clause.
* [HQL Where Clause Example](http://www.roseindia.net/hibernate/hqlwhereclauseexample.shtml)  
  Where Clause is used to limit the results returned from database. In this lesson you will learn how to use HQL Where Clause.
* [HQL Group By Clause Example](http://www.roseindia.net/hibernate/groupbyclauseexample.shtml)  
  Group by clause is used to return the aggregate values by grouping on returned component. In this lesson you will learn how to use HQL Group By Clause.
* [HQL Order By Example](http://www.roseindia.net/hibernate/hibernateorderbyexample.shtml)  
  Order by clause is used to retrieve the data from database in the sorted order by any property of returned class or components. In this lesson you will learn how to use HQL Order By Clause.  
      
     
  **Hibernate Criteria Query**
* [Hibernate Criteria Query Example](http://www.roseindia.net/hibernate/criteriaqueyexample.shtml)  
  In this lesson you will learn how to use Hibernate Criteria Query.
* [Narrowing the result set](http://www.roseindia.net/hibernate/criteriaqueries.shtml)  
  In this lesson you will learn how to use Hibernate Criteria Query to narrow down the result data.

**What is Hibernate?**

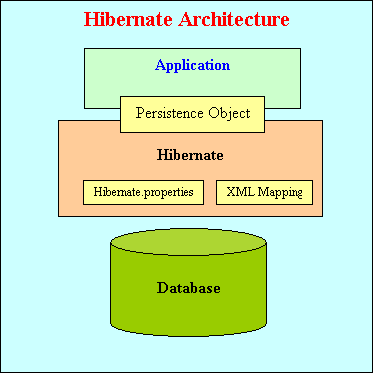
Hibernate 3.0, the latest Open Source persistence technology at the heart of J2EE EJB 3.0 is available for download from [Hibernet.org](http://www.hibernate.org/Download/DownloadOverview).The Hibernate 3.0 core is 68,549 lines of Java code together with 27,948 lines of unit tests, all freely available under the LGPL, and has been in development for well over a year. 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.
* Filters for working with temporal (historical), regional or permissioned data.
* Enhanced Criteria query API: with full support for projection/aggregation and subselects.
* Runtime performance monitoring: via JMX or local Java API, including a second-level cache browser.
* 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.
* JDK 1.5 Enhancements: The new JDK has been released as a preview earlier this year and we expect a slow migration to the new 1.5 platform throughout 2004. While Hibernate3 still runs perfectly with JDK 1.2, Hibernate3 will make use of some new JDK features. JSR 175 annotations, for example, are a perfect fit for Hibernate metadata and we will embrace them aggressively. We will also support Java generics, which basically boils down to allowing type safe collections.
* 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

In this lesson you will learn the architecture of Hibernate.  The following diagram describes the high level architecture of hibernate:



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.

# Writing First Hibernate Code

In this section I will show you how to create a simple program to insert record in MySQL database. You can run this program from Eclipse or from command prompt as well. I am assuming that you are familiar with MySQL and Eclipse environment.

**Configuring Hibernate**  
In this application Hibernate provided connection pooling and transaction management is used for simplicity. Hibernate uses the hibernate.cfg.xml to create the connection pool and setup required environment.

**Here is the code:**

|  |
| --- |
| <?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:

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  /\*\*  \* @author Deepak Kumar  \*  \* Java Class to map to the datbase Contact Table  \*/ **public class**Contact {   **private**String firstName;   **private**String lastName;   **private**String email;   **private long**id;    /\*\*    \* @return Email    \*/   **public**String getEmail() {     **return**email;   }    /\*\*    \* @return First Name    \*/   **public**String getFirstName() {     **return**firstName;   }    /\*\*     \* @return Last name    \*/   **public**String getLastName() {     **return**lastName;   }    /\*\*    \* @param string Sets the Email    \*/   **public void**setEmail(String string) {     email = string;   }    /\*\*    \* @param string Sets the First Name    \*/   **public void**setFirstName(String string) {     firstName = string;   }    /\*\*    \* @param string sets the Last Name    \*/   **public void**setLastName(String string) {     lastName = string;   }    /\*\*    \* @return ID Returns ID    \*/   **public long**getId() {     **return**id;   }    /\*\*    \* @param l Sets the ID    \*/   **public void**setId(**long**l) {     id = l;   }  } |

**Mapping the Contact Object to the Database Contact table**  
The file contact.hbm.xml is used to map Contact Object to the Contact table in the database. Here is the code for contact.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?> <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>   <class name="roseindia.tutorial.hibernate.Contact" table="CONTACT">    <id name="id" type="long" column="ID" >    <generator class="assigned"/>   </id>    <property name="firstName">      <column name="FIRSTNAME" />   </property>   <property name="lastName">     <column name="LASTNAME"/>   </property>   <property name="email">     <column name="EMAIL"/>   </property>  </class> </hibernate-mapping> |

**Setting Up MySQL Database**  
In the configuration file(hibernate.cfg.xml) we have specified to use **hibernatetutorial** database running on localhost.  So, create the databse ("**hibernatetutorial")** on the MySQL server running on localhost.

**Developing Code to Test Hibernate example**  
Now we are ready to write a program to insert the data into database. We should first understand about the Hibernate's Session. 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:

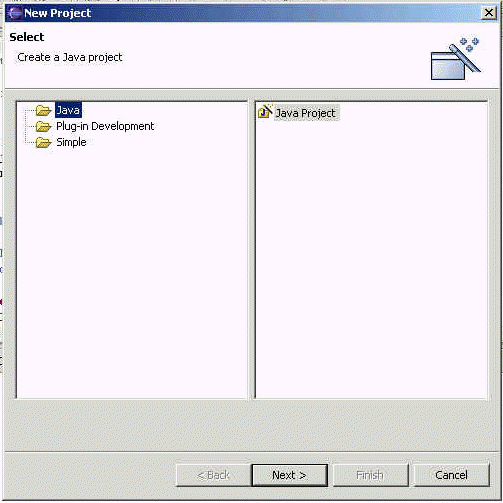
**session.save(contact)**

Here is the code of FirstExample.java

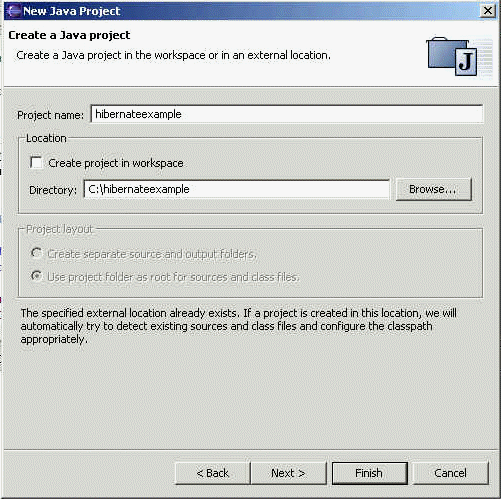
|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.SessionFactory; **import**org.hibernate.cfg.Configuration;   /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* Hibernate example to inset data into Contact table  \*/ **public class**FirstExample {   **public static void**main(String[] args) {     Session session = **null**;      **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for use       SessionFactory sessionFactory = **new**Configuration().configure().buildSessionFactory();        session =sessionFactory.openSession();         //Create new instance of Contact and set values in it by reading them from form object          System.out.println("Inserting Record");         Contact contact = **new**Contact();         contact.setId(3);         contact.setFirstName("Deepak");         contact.setLastName("Kumar");         contact.setEmail("deepak\_38@yahoo.com");         session.save(contact);         System.out.println("Done");     }**catch**(Exception e){       System.out.println(e.getMessage());     }**finally**{       // Actual contact insertion will happen at this step       session.flush();       session.close();        }        } } |

# Running First Hibernate 3.0 Example

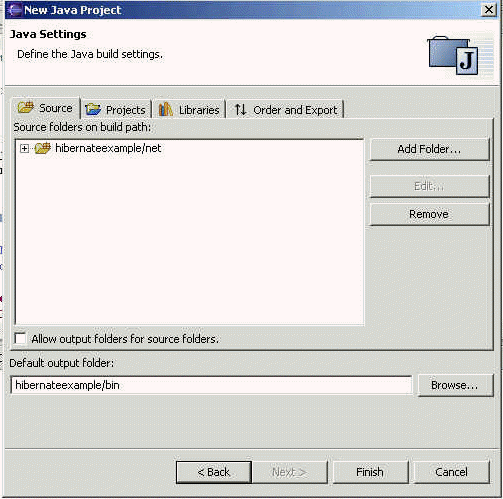
Hibernate is free open source software it can be download from <http://www.hibernate.org/6.html>. Visit the site and download Hibernate 3.0. You can download the Hibernate and install it yourself. But I have provided very thing in one zip file. Download the example code and library from [here](http://www.roseindia.net/hibernate/hibernate.zip) and extract the content in your favorite directory say "C:\hibernateexample". Download file contains the Eclipse project. To run the example you should have the Eclipse IDE on your machine. Start the Eclipse project and select Java Project as shown below.



Click on "Next" button. In the new screen, enter "hibernateexample" as project name and browse the extracted directory "C:\hibernateexample".

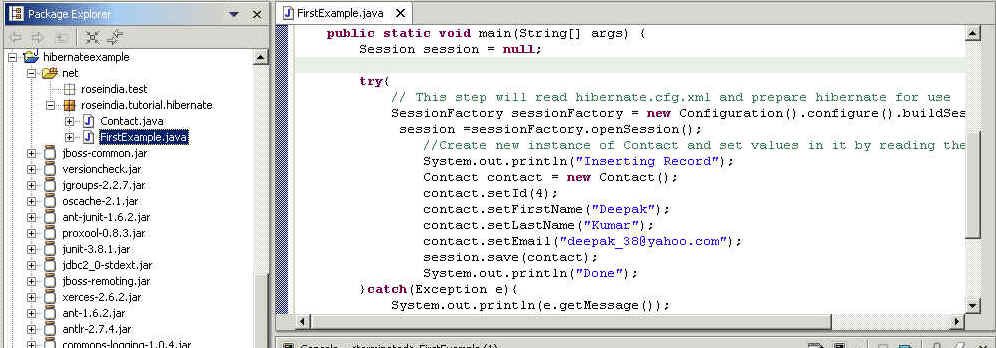


Click on "Next" button. In the next screen leave the output folder as default "hibernateexample/bin" .

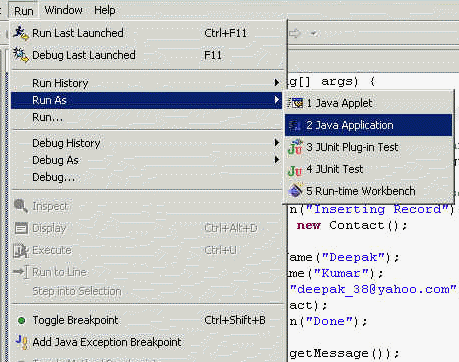


Click on the "Finish" button.

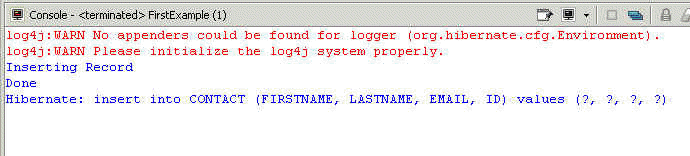
Now Open the FirstExample.java in the editor as show below.



Copy  contact.hbm.xml, and hibernate.cfg.xml in the bin directory of the project using windows explorer. To run the example select Run-> Run As -> Java Application from the menu bar as shown below.



This will run the Hibernate example program in Eclipse following output will displayed on the Eclipse Console.



In this section I showed you how to run the our first Hibernate 3.0 example.

# Understanding Hibernate O/R Mapping

In the last example we created contact.hbm.xml to map Contact Object to the Contact table in the database. Now let's understand the each component of the mapping file.

To recall here is the content of contact.hbm.xml:

|  |
| --- |
| <?xml version="1.0"?> <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>   <class name="roseindia.tutorial.hibernate.Contact" table="CONTACT">    <id name="id" type="long" column="ID" >    <generator class="assigned"/>   </id>    <property name="firstName">      <column name="FIRSTNAME" />   </property>   <property name="lastName">     <column name="LASTNAME"/>   </property>   <property name="email">     <column name="EMAIL"/>   </property>  </class> </hibernate-mapping> |

Hibernate mapping documents are simple xml documents. Here are important elements of the mapping file:.

1. **<hibernate-mapping> element**  
   The first or root element of hibernate mapping document is <hibernate-mapping> element. Between the <**hibernate-mapping>** tag class element(s) are present.
2. **<class> element**  
   The <Class> element maps the class object with corresponding entity in the database. It also tells what table in the database has to access and what column in that table it should use. Within one <hibernate-mapping> element, several <class> mappings are possible.
3. **<id> element**  
   The <id> element in unique identifier to identify and object. In fact <id> element map with the primary key of the table. In our code :  
   <id name="id" type="long" column="ID" >  
   primary key maps to the **ID** field of the table **CONTACT**. The attributes of the id element are:
   * **name:** The property name used by the persistent class.
   * **column:** The column used to store the primary key value.
   * **type:** The Java data type used.
   * **unsaved-value:** This is the value used to determine if a class has been made persistent. If the value of the id attribute is null, then it means that this object has not been persisted.
4. **<generator> element**  
   The **<generator>** methodis used to generate the primary key for the new record. Here is some of the commonly used generators :  
        
   **\*** **Increment** - This is used to generate primary keys of type long, short or int that are unique only. It should not be used in the clustered deployment environment.  
        
   **\*  Sequence** - Hibernate can also use the sequences to generate the primary key. It can be used with DB2, PostgreSQL, Oracle, SAP DB databases.  
       
   **\* Assigned** - Assigned method is used when application code generates the primary key.
5. **<property> element**  
   The property elements define standard Java attributes and their mapping into database schema. The property element supports the column child element to specify additional properties, such as the index name on a column or a specific column type.

# 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.

**The <generator> element**

This is the optional element under <id> element. The <generator> element is used to specify the class name to be used to generate the primary key for new record while saving a new record. The <param> element is used to pass the parameter (s) to the  class. Here is the example of generator element from our first application:  
**<generator class="assigned"/>**In this case <generator> element do not generate the primary key and it is required to set the primary key value before calling save() method.

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

|  |  |
| --- | --- |
| **Generator** | **Description** |
| increment | It generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. It should not the used in the clustered environment. |
| identity | It supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL. The returned identifier is of type long, short or int. |
| sequence | The sequence generator uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or a generator in Interbase. The returned identifier is of type long, short or int |
| hilo | The hilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a table and column (by default hibernate\_unique\_key and next\_hi respectively) as a source of hi values. The hi/lo algorithm generates identifiers that are unique only for a particular database. Do not use this generator with connections enlisted with JTA or with a user-supplied connection. |
| seqhilo | The seqhilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a named database sequence. |
| uuid | The uuid generator uses a 128-bit UUID algorithm to generate identifiers of type string, unique within a network (the IP address is used). The UUID is encoded as a string of hexadecimal digits of length 32. |
| guid | It uses a database-generated GUID string on MS SQL Server and MySQL. |
| native | It picks identity, sequence or hilo depending upon the capabilities of the underlying database. |
| assigned | lets the application to assign an identifier to the object before save() is called. This is the default strategy if no <generator> element is specified. |
| select | retrieves a primary key assigned by a database trigger by selecting the row by some unique key and retrieving the primary key value. |
| foreign | uses the identifier of another associated object. Usually used in conjunction with a <one-to-one> primary key association. |

# Using Hibernate <generator> to generate id incrementally

As we have seen in the last section that the increment class generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. In this lesson I will show you how to write running program to demonstrate it. You should not use this method to generate the primary key in case of clustured environment.

In this we will create a new table in database, add mappings in the contact.hbm.xml file, develop the POJO class (Book.java), write the program to test it out.

**Create Table in the mysql database:**User the following sql statement to create a new table in the database.  
 CREATE TABLE `book` (   
`id` int(11) NOT NULL default '0',   
`bookname` varchar(50) default NULL,   
PRIMARY KEY (`id`)   
) TYPE=MyISAM

**Developing POJO Class (Book.java)**  
Book.java is our POJO class which is to be persisted to the database table "book".

|  |
| --- |
| /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* Java Class to map to the database Book table  \*/ **package**roseindia.tutorial.hibernate;   **public class**Book {   **private long**lngBookId;   **private**String strBookName;      /\*\*    \* @return Returns the lngBookId.    \*/   **public long**getLngBookId() {     **return**lngBookId;   }   /\*\*    \* @param lngBookId The lngBookId to set.    \*/   **public void**setLngBookId(**long**lngBookId) {     **this**.lngBookId = lngBookId;   }   /\*\*    \* @return Returns the strBookName.    \*/   **public**String getStrBookName() {     **return**strBookName;   }   /\*\*    \* @param strBookName The strBookName to set.    \*/   **public void**setStrBookName(String strBookName) {     **this**.strBookName = strBookName;   } } |

Adding Mapping entries to contact.hbm.xml

|  |
| --- |
| <class name="roseindia.tutorial.hibernate.Book" table="book">      <id name="lngBookId" type="long" column="id" >         **<generator class="increment"/>**      </id>      <property name="strBookName">          <column name="bookname" />      </property> </class> |

 Note that we have used **increment** for the generator class. \*After adding the entries to the xml file copy it to the bin directory of your hibernate eclipse project(this step is required if you are using eclipse).

**Write the client program and test it out**  
Here is the code of our client program to test the application.

|  |
| --- |
| /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* Example to show the increment class of hibernate generator element to   \* automatically generate the primay key  \*/ **package**roseindia.tutorial.hibernate;  //Hibernate Imports **import**org.hibernate.Session; **import**org.hibernate.SessionFactory; **import**org.hibernate.cfg.Configuration;   **public class**IdIncrementExample {   **public static void**main(String[] args) {     Session session = **null**;      **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for use       SessionFactory sessionFactory = **new**Configuration().configure().buildSessionFactory();       session =sessionFactory.openSession();               org.hibernate.Transaction tx = session.beginTransaction();               //Create new instance of Contact and set values in it by reading them from form object        System.out.println("Inserting Book object into database..");       Book book = **new**Book();       book.setStrBookName("Hibernate Tutorial");       session.save(book);       System.out.println("Book object persisted to the database.");           tx.commit();           session.flush();           session.close();     }**catch**(Exception e){       System.out.println(e.getMessage());     }**finally**{       }        } } |

To test the program Select Run->Run As -> Java Application from the eclipse menu bar. This will create a new record into the book table.

# 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. In this lesson we will understand HQL in detail.

**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).

**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

**Aggregate functions are:**

* avg(...), sum(...), min(...), max(...)
* count(\*)
* count(...), count(distinct ...), count(all...)

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

# Preparing table for HQL Examples

In this lesson we will create insurance table and populate it with the data. We will use insurance table for rest of the HQL tutorial.

To create the insurance table and insert the sample data, run the following sql query:

|  |
| --- |
| /\*Table structure for table `insurance` \*/  drop table if exists `insurance`;  CREATE TABLE `insurance` (  `ID` int(11) NOT NULL default '0',  `insurance\_name` varchar(50) default NULL,  `invested\_amount` int(11) default NULL,  `investement\_date` datetime default NULL,  PRIMARY KEY (`ID`)  ) TYPE=MyISAM;  /\*Data for the table `insurance` \*/  insert into `insurance` values (1,'Car Insurance',1000,'2005-01-05 00:00:00');  insert into `insurance` values (2,'Life Insurance',100,'2005-10-01 00:00:00');  insert into `insurance` values (3,'Life Insurance',500,'2005-10-15 00:00:00');  insert into `insurance` values (4,'Car Insurance',2500,'2005-01-01 00:00:00');  insert into `insurance` values (5,'Dental Insurance',500,'2004-01-01 00:00:00');  insert into `insurance` values (6,'Life Insurance',900,'2003-01-01 00:00:00');  insert into `insurance` values (7,'Travel Insurance',2000,'2005-02-02 00:00:00');  insert into `insurance` values (8,'Travel Insurance',600,'2005-03-03 00:00:00');  insert into `insurance` values (9,'Medical Insurance',700,'2005-04-04 00:00:00');  insert into `insurance` values (10,'Medical Insurance',900,'2005-03-03 00:00:00');  insert into `insurance` values (11,'Home Insurance',800,'2005-02-02 00:00:00');  insert into `insurance` values (12,'Home Insurance',750,'2004-09-09 00:00:00');  insert into `insurance` values (13,'Motorcycle Insurance',900,'2004-06-06 00:00:00');  insert into `insurance` values (14,'Motorcycle Insurance',780,'2005-03-03 00:00:00'); |

Above Sql query will create insurance table and add the following data:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **insurance\_name** | **invested\_amount** | **investement\_date** |
| 1 | Car Insurance | 1000 | 2005-01-05 00:00:00 |
| 2 | Life Insurance | 100 | 2005-10-01 00:00:00 |
| 3 | Life Insurance | 500 | 2005-10-15 00:00:00 |
| 4 | Car Insurance | 2500 | 2005-01-01 00:00:00 |
| 5 | Dental Insurance | 500 | 2004-01-01 00:00:00 |
| 6 | Life Insurance | 900 | 2003-01-01 00:00:00 |
| 7 | Travel Insurance | 2000 | 2005-02-02 00:00:00 |
| 8 | Travel Insurance | 600 | 2005-03-03 00:00:00 |
| 9 | Medical Insurance | 700 | 2005-04-04 00:00:00 |
| 10 | Medical Insurance | 900 | 2005-03-03 00:00:00 |
| 11 | Home Insurance | 800 | 2005-02-02 00:00:00 |
| 12 | Home Insurance | 750 | 2004-09-09 00:00:00 |
| 13 | Motorcycle Insurance | 900 | 2004-06-06 00:00:00 |
| 14 | Motorcycle Insurance | 780 | 2005-03-03 00:00:00 |

In the future lessons we will use this table to write different HQL examples.

# Writing ORM for Insurance table

In this lesson we will write the java class and add necessary code in the contact.hbm.xml file.

**Create POJO class:**Here is the code of our java file (Insurance.java), which we will map to the insurance table.

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**java.util.Date; /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* Java Class to map to the database insurance table  \*/ **public class**Insurance {   **private long**lngInsuranceId;   **private**String insuranceName;   **private int**investementAmount;   **private**Date investementDate;      /\*\*    \* @return Returns the insuranceName.    \*/   **public**String getInsuranceName() {     **return**insuranceName;   }   /\*\*    \* @param insuranceName The insuranceName to set.    \*/   **public void**setInsuranceName(String insuranceName) {     **this**.insuranceName = insuranceName;   }   /\*\*    \* @return Returns the investementAmount.    \*/   **public int**getInvestementAmount() {     **return**investementAmount;   }   /\*\*    \* @param investementAmount The investementAmount to set.    \*/   **public void**setInvestementAmount(**int**investementAmount) {     **this**.investementAmount = investementAmount;   }   /\*\*    \* @return Returns the investementDate.    \*/   **public**Date getInvestementDate() {     **return**investementDate;   }   /\*\*    \* @param investementDate The investementDate to set.    \*/   **public void**setInvestementDate(Date investementDate) {     **this**.investementDate = investementDate;   }   /\*\*    \* @return Returns the lngInsuranceId.    \*/   **public long**getLngInsuranceId() {     **return**lngInsuranceId;   }   /\*\*    \* @param lngInsuranceId The lngInsuranceId to set.    \*/   **public void**setLngInsuranceId(**long**lngInsuranceId) {     **this**.lngInsuranceId = lngInsuranceId;   } } |

**Adding mappings into  contact.hbm.xml file**  
Add the following code into  contact.hbm.xml file.

|  |
| --- |
| <class name="roseindia.tutorial.hibernate.Insurance" table="insurance">  <id name="lngInsuranceId" type="long" column="ID" >  <generator class="increment"/>  </id>  <property name="insuranceName">  <column name="insurance\_name" />  </property>  <property name="investementAmount">  <column name="invested\_amount" />  </property>  <property name="investementDate">  <column name="investement\_date" />  </property>  </class> |

Now we have created the POJO class and necessary mapping into contact.hbm.xml file.

# HQL from clause Example

In this example you will learn how to use the HQL from clause. The from clause is the simplest possible Hibernate Query. Example of from clause is:

from Insurance insurance

Here is the full code of the from clause example:

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*;  **import**java.util.\*;  /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* Select HQL Example  \*/ **public class**SelectHQLExample {    **public static void**main(String[] args) {   Session session = **null**;    **try**{     // This step will read hibernate.cfg.xml and prepare hibernate for use     SessionFactory sessionFactory = **new**Configuration().configure().buildSessionFactory();      session =sessionFactory.openSession();                   //Using from Clause        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();   }**catch**(Exception e){     System.out.println(e.getMessage());   }**finally**{     }    }   } |

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

|  |
| --- |
| log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).  log4j:WARN Please initialize the log4j system properly.  Hibernate: select insurance0\_.ID as col\_0\_0\_ from insurance insurance0\_  ID: 1  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Car Insurance  ID: 2  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Life Insurance  ID: 3  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Life Insurance  ID: 4  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Car Insurance  ID: 5  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Dental Insurance  ID: 6  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Life Insurance  ID: 7  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Travel Insurance  ID: 8  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Travel Insurance  ID: 9  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Medical Insurance  ID: 10  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Medical Insurance  ID: 11  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Home Insurance  ID: 12  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Home Insurance  ID: 13  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Motorcycle Insurance  ID: 14  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  First Name: Motorcycle Insurance |

# Hibernate Select Clause

In this lesson we will write example code to select the data from Insurance table using Hibernate Select Clause. The select clause picks up objects and properties to return in the query result set. Here is the query:

Select insurance.lngInsuranceId, insurance.insuranceName, insurance.investementAmount, insurance.investementDate from Insurance insurance

which selects all the rows (**insurance.lngInsuranceId, insurance.insuranceName, insurance.investementAmount, insurance.investementDate**) from Insurance table.

Hibernate generates the necessary sql query and selects all the records from Insurance table. Here is the code of our java file which shows how select HQL can be used:

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*;  **import**java.util.\*;  /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* HQL Select Clause Example  \*/ **public class**SelectClauseExample {   **public static void**main(String[] args) {   Session session = **null**;    **try**{     // This step will read hibernate.cfg.xml and prepare hibernate for use     SessionFactory sessionFactory = **new**Configuration().configure() .buildSessionFactory();     session =sessionFactory.openSession();           //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();   }**catch**(Exception e){     System.out.println(e.getMessage());   }**finally**{     }   } } |

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

|  |
| --- |
| Hibernate: select insurance0\_.ID as col\_0\_0\_, insurance0\_.insurance\_name as col\_1\_0\_, insurance0\_.invested\_amount as col\_2\_0\_, insurance0\_.investement\_date as col\_3\_0\_ from insurance insurance0\_  ID: 1  Name: Car Insurance  Amount: 1000  ID: 2  Name: Life Insurance  Amount: 100  ID: 3  Name: Life Insurance  Amount: 500  ID: 4  Name: Car Insurance  Amount: 2500  ID: 5  Name: Dental Insurance  Amount: 500  ID: 6  Name: Life Insurance  Amount: 900  ID: 7  Name: Travel Insurance  Amount: 2000  ID: 8  Name: Travel Insurance  Amount: 600  ID: 9  Name: Medical Insurance  Amount: 700  ID: 10  Name: Medical Insurance  Amount: 900  ID: 11  Name: Home Insurance  Amount: 800  ID: 12  Name: Home Insurance  Amount: 750  ID: 13  Name: Motorcycle Insurance  Amount: 900  ID: 14  Name: Motorcycle Insurance  Amount: 780 |

# HQL Where Clause Example

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. For example:

from Insurance where lngInsuranceId='1'

Where Clause can be used with or without Select Clause. Here the example code:

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*;  **import**java.util.\*;  /\*\*  \* @author Deepak Kumar  \*  \* http://www.roseindia.net  \* HQL Where Clause Example  \* Where Clause With Select Clause Example  \*/ **public class**WhereClauseExample {   **public static void**main(String[] args) {   Session session = **null**;    **try**{     // This step will read hibernate.cfg.xml and prepare hibernate for use     SessionFactory sessionFactory = **new**Configuration().configure().buildSessionFactory();     session =sessionFactory.openSession();             System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");       System.out.println("Query using Hibernate Query Language");     //Query using Hibernate Query Language      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());              }      System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");      System.out.println("Where Clause With Select Clause");     //Where Clause With Select Clause      SQL\_QUERY ="Select insurance.lngInsuranceId,insurance.insuranceName," +      "insurance.investementAmount,insurance.investementDate from Insurance insurance "+      " where insurance.lngInsuranceId='1'";      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("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          session.close();   }**catch**(Exception e){     System.out.println(e.getMessage());   }**finally**{     }       } } |

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Query using Hibernate Query Language  Hibernate: select insurance0\_.ID as col\_0\_0\_ from insurance insurance0\_ where (insurance0\_.ID='1')  ID: 1  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Car Insurance  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Where Clause With Select Clause  Hibernate: select insurance0\_.ID as col\_0\_0\_, insurance0\_.insurance\_name as col\_1\_0\_, insurance0\_.invested\_amount as col\_2\_0\_, insurance0\_.investement\_date as col\_3\_0\_ from insurance insurance0\_ where (insurance0\_.ID='1')  ID: 1  Name: Car Insurance  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

**HQL Group By Clause Example**

Group by clause is used to return the aggregate values by grouping on returned component. HQL supports Group By Clause. In our example we will calculate the sum of invested amount in each insurance type. Here is the java code for calculating the invested amount insurance wise:

|  |
| --- |
| **package**roseindia.tutorial.hibernate; **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*; **import**java.util.\*; /\*\*  \* @author Deepak Kumar  \*   \* http://www.roseindia.net HQL Group by Clause Example  \*    \*/ **public class**HQLGroupByExample {   **public static void**main(String[] args) {     Session session = **null**;     **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for       // use       SessionFactory sessionFactory = **new**Configuration().configure()           .buildSessionFactory();       session = sessionFactory.openSession();       //Group By Clause Example       String SQL\_QUERY = "select sum(insurance.investementAmount),insurance.insuranceName "           + "from Insurance insurance group by insurance.insuranceName";       Query query = session.createQuery(SQL\_QUERY);       **for**(Iterator it = query.iterate(); it.hasNext();) {         Object[] row = (Object[]) it.next();         System.out.println("Invested Amount: " + row[0]);         System.out.println("Insurance Name: " + row[1]);       }       session.close();     } **catch**(Exception e) {       System.out.println(e.getMessage());     } **finally**{     }   } } |

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

|  |
| --- |
| Hibernate: select sum(insurance0\_.invested\_amount) as col\_0\_0\_, insurance0\_.insurance\_name as col\_1\_0\_ from insurance insurance0\_ group by insurance0\_.insurance\_name  Invested Amount: 3500  Insurance Name: Car Insurance  Invested Amount: 500  Insurance Name: Dental Insurance  Invested Amount: 1550  Insurance Name: Home Insurance  Invested Amount: 1500  Insurance Name: Life Insurance  Invested Amount: 1600  Insurance Name: Medical Insurance  Invested Amount: 1680  Insurance Name: Motorcycle Insurance  Invested Amount: 2600  Insurance Name: Travel Insurance |

# HQL Order By Example

Order by clause is used to retrieve the data from database in the sorted order by any property of returned class or components. HQL supports Order By Clause. In our example we will retrieve the data sorted on the insurance type. Here is the java example code:

|  |
| --- |
| **package**roseindia.tutorial.hibernate; **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*; **import**java.util.\*; /\*\*  \* @author Deepak Kumar  \*   \* http://www.roseindia.net HQL Order by Clause Example  \*    \*/ **public class**HQLOrderByExample {   **public static void**main(String[] args) {     Session session = **null**;     **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for       // use       SessionFactory sessionFactory = **new**Configuration().configure()           .buildSessionFactory();       session = sessionFactory.openSession();       //Order By Example       String SQL\_QUERY = " from Insurance as insurance order by insurance.insuranceName";       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());       }       session.close();     } **catch**(Exception e) {       System.out.println(e.getMessage());     } **finally**{     }   } } |

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

|  |
| --- |
| Hibernate: select insurance0\_.ID as col\_0\_0\_ from insurance insurance0\_ order by insurance0\_.insurance\_name  ID: 1  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Car Insurance  ID: 4  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Car Insurance  ID: 5  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Dental Insurance  ID: 11  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Home Insurance  ID: 12  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Home Insurance  ID: 2  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Life Insurance  ID: 3  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Life Insurance  ID: 6  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Life Insurance  ID: 9  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Medical Insurance  ID: 10  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Medical Insurance  ID: 13  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Motorcycle Insurance  ID: 14  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Motorcycle Insurance  ID: 7  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Travel Insurance  ID: 8  Hibernate: select insurance0\_.ID as ID0\_, insurance0\_.insurance\_name as insurance2\_2\_0\_, insurance0\_.invested\_amount as invested3\_2\_0\_, insurance0\_.investement\_date as investem4\_2\_0\_ from insurance insurance0\_ where insurance0\_.ID=?  Name: Travel Insurance |

# 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. Here is a simple example of Hibernate Criterial Query:

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.cfg.\*; **import**java.util.\*; /\*\*  \* @author Deepak Kumar  \*   \* http://www.roseindia.net Hibernate Criteria Query Example  \*    \*/**public class**HibernateCriteriaQueryExample {   **public static void**main(String[] args) {     Session session = **null**;     **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for       // use       SessionFactory sessionFactory = **new**Configuration().configure()           .buildSessionFactory();       session = sessionFactory.openSession();       //Criteria Query Example       Criteria crit = session.createCriteria(Insurance.**class**);       List insurances = crit.list();       **for**(Iterator it = insurances.iterator();it.hasNext();){         Insurance insurance = (Insurance) it.next();         System.out.println("ID: " + insurance.getLngInsuranceId());         System.out.println("Name: " + insurance.getInsuranceName());                }       session.close();     } **catch**(Exception e) {       System.out.println(e.getMessage());     } **finally**{     }       } } |

The above Criteria Query example selects all the records from the table and displays on the console. In the above code the following code creates a new Criteria instance, for the class **Insurance**:

Criteria crit = session.createCriteria(Insurance.class);

The code:

List insurances = crit.list();

creates the sql query and execute against database to retrieve the data.

# Criteria Query Examples

In the last lesson we learnt how to use Criteria Query to select all the records from Insurance table. In this lesson we will learn how to restrict the results returned from the database. 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 |

Here is an example code that shows how to use Restrictions.like method and restrict the maximum rows returned by query by setting the Criteria.setMaxResults() value to 5.

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.criterion.\*; **import**org.hibernate.cfg.\*; **import**java.util.\*; /\*\*  \* @author Deepak Kumar  \*   \* http://www.roseindia.net Hibernate Criteria Query Example  \*    \*/**public class**HibernateCriteriaQueryExample2 {   **public static void**main(String[] args) {     Session session = **null**;     **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for       // use       SessionFactory sessionFactory = **new**Configuration().configure()           .buildSessionFactory();       session = sessionFactory.openSession();       //Criteria Query Example       Criteria crit = session.createCriteria(Insurance.**class**);       crit.add(Restrictions.like("insuranceName", "%a%")); //Like condition       crit.setMaxResults(5); //Restricts the max rows to 5        List insurances = crit.list();       **for**(Iterator it = insurances.iterator();it.hasNext();){         Insurance insurance = (Insurance) it.next();         System.out.println("ID: " + insurance.getLngInsuranceId());         System.out.println("Name: " + insurance.getInsuranceName());                }       session.close();     } **catch**(Exception e) {       System.out.println(e.getMessage());     } **finally**{     }       } } |

**Hibernate Native SQL Example**

**Native SQL** is handwritten SQL for all database operations like create, update, delete and select. **Hibernate Native Query** also supports stored procedures. Hibernate allows you to run Native SQL Query for all the database operations, so you can use your existing handwritten sql with Hibernate, this also helps you in migrating your SQL/JDBC based application to Hibernate.

In this example we will show you how you can use Native SQL with hibernate. You will learn how to use Native to calculate average and then in another example select all the objects from table.

**Here is the code of Hibernate Native SQL:**

|  |
| --- |
| **package**roseindia.tutorial.hibernate;  **import**org.hibernate.Session; **import**org.hibernate.\*; **import**org.hibernate.criterion.\*; **import**org.hibernate.cfg.\*; **import**java.util.\*; /\*\*  \* @author Deepak Kumar  \*   \* http://www.roseindia.net Hibernate Native Query Example  \*    \*/ **public class**NativeQueryExample {   **public static void**main(String[] args) {     Session session = **null**;      **try**{       // This step will read hibernate.cfg.xml and prepare hibernate for use       SessionFactory sessionFactory = **new**Configuration().configure().buildSessionFactory();       session =sessionFactory.openSession();       /\* Hibernate Native Query Average Examle\*/        String sql ="select stddev(ins.invested\_amount) as stdErr, "+          " avg(ins.invested\_amount) as mean "+          " from insurance ins";        Query query = session.createSQLQuery(sql).addScalar("stdErr",Hibernate.DOUBLE).          addScalar("mean",Hibernate.DOUBLE);        //Double [] amount = (Double []) query.uniqueResult();         Object [] amount = (Object []) query.uniqueResult();         System.out.println("mean amount: " + amount[0]);        System.out.println("stdErr amount: " + amount[1]);         /\* Example to show Native query to select all the objects from database \*/        /\* Selecting all the objects from insurance table \*/        List insurance = session.createSQLQuery("select  {ins.\*}  from insurance ins")       .addEntity("ins", Insurance.**class**)         .list();       **for**(Iterator it = insurance.iterator(); it.hasNext();) {         Insurance insuranceObject = (Insurance) it.next();         System.out.println("ID: " + insuranceObject.getLngInsuranceId());         System.out.println("Name: " + insuranceObject.getInsuranceName());       }                  session.close();     }**catch**(Exception e){       System.out.println(e.getMessage());       e.printStackTrace();     }        } } |

Following query is used to calculate the average of  invested amount:

/\*Hibernate Native Query Average Examle\*/

String sql ="select stddev(ins.invested\_amount) as stdErr, "+ " avg(ins.invested\_amount) as mean "+ " from insurance ins";

The following code:

Query query = session.createSQLQuery(sql).addScalar("stdErr",Hibernate.DOUBLE).  
addScalar("mean",Hibernate.DOUBLE);

Creates a new instance of SQLQuery for the given SQL query string and the entities returned by the query are detached.

To return all the entities from database we have used the following query:

/\* Example to show Native query to select all the objects from database \*/  
/\* Selecting all the objects from insurance table \*/  
List insurance = session.createSQLQuery("**select {ins.\*} from insurance ins**")  
.addEntity("**ins**", **Insurance.class**)  
.list();  
    for (Iterator it = insurance.iterator(); it.hasNext();) {  
    Insurance insuranceObject = (Insurance) it.next();  
    System.out.println("ID: " + insuranceObject.getLngInsuranceId());  
   System.out.println("Name: " + insuranceObject.getInsuranceName());  
}

When you run the program through it should display the following result:

|  |
| --- |
| log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).  log4j:WARN Please initialize the log4j system properly.  Hibernate: select stddev(ins.invested\_amount) as stdErr, avg(ins.invested\_amount) as mean from insurance ins  mean amount: 592.1584  stdErr amount: 923.5714  Hibernate: select ins.ID as ID0\_, ins.insurance\_name as insurance2\_2\_0\_, ins.invested\_amount as invested3\_2\_0\_, ins.investement\_date as investem4\_2\_0\_ from insurance ins  ID: 1  Name: Car Insurance  ID: 2  Name: Life Insurance  ID: 3  Name: Life Insurance  ID: 4  Name: Car Insurance  ......  ....... |

In this example you learned how to use Native Query with Hibernate.

# hibernateTemplate-Order Results

In this section you will learn how to add ordering ability to in Hibernate Application when using hibernateTemplate. There are many ways you can order your results when using hibernateTemplate.

Suppose you have a class called "**Student**" with the fields "**studentName**", "**StudentCode**" and you want to retrieve all the records from database in sorted by "**studentName**" in ascending order. There are following options for you.

1. **Use getHibernateTemplate().find() method**  
   Following code shows how you can use getHibernateTemplate().find() method:  
   **public** Collection loadAllStudent() **throws** DataAccessException{  
       **return** getHibernateTemplate().find("from **Student** student order by **student.studentName**");  
   }
2. **Use getHibernateTemplate().findByCriteria()**  
   Following code shows how you can use getHibernateTemplate().findByCriteria() function in your hibernate application.  
   **public** Collection loadAllStudent() **throws** DataAccessException{  
   **return** getHibernateTemplate().findByCriteria(DetachedCriteria.forClass(Student.**class**).addOrder(Order.asc("studentName")));  
   }

# Hibernate Types

This section gives you description of all the Types that are supported by Hibernate. A Hibernate Type is used to map a Java property type to a JDBC type or types.

The following tables to represents all Hibernate types:

|  |  |
| --- | --- |
| **Interfaces and Descriptions:** | |
| **AbstractComponentType** | The AbstractComponentType enables other Component-like types to hold collections and have cascades, etc. |
| **AssociationType** | This interface used to represent all associations between entities. |
| **DiscriminatorType** | This interface used to discriminator properties with the help of right mapped subclass. |
| **IdentifierType** | This interface has All identifiers of entities. |
| **LiteralType** | This is a maker interface that which store SQL literals. |
| **Type** | This interface describes mapping between the Java and JDBC datatypes |
| **VersionType** | This interface used for version stamping. |

|  |  |
| --- | --- |
| **Classes and Descriptions:** | |
| **AbstractBynaryType** | The stream of byte bounded into a VAQRBINARY. |
| **AbstractCharArrayType** | The stream of char bounded into a VARCHAR. |
| **AbstractType** | This is a superclass that can be used for creating type hierarchy. |
| **AdaptedImmutableType** |  |
| **AnyType** | It defines 'any' mappings and deprecated 'object' types. |
| **AnyType.ObjectTypeCacheEntry** |  |
| **ArrayType** | It represents collection of data into a similar types. |
| **BagType** |  |
| **BigDecimaType** | This class used to mapping between SQL NUMERIC and java.math.BigDecimal. |
| **BigIntegerType** | This class used to mapping between SQL NUMERIC and java.math.BigInteger. |
| **BinaryType** | It used to mapping between a SQL VARBINARY and a Java byte[]. |
| **BlobType** | It used to mapping between a SQL BLOB and java.sql.Blob. |
| **BooleanType** | This class maps between SQL BIT and Java Boolean. |
| **ByteType** | This class maps between SQL TINYINT  and Java Byte. |
| **CalendarDateType** | This class represents a data and mapping into a Calendar object. |
| **CalendarType** | It also represents a datetime mapping into a Calendar object. |
| **CharacterArrayType** | It is a collection of VARCHAR like: Character[]. |
| **CharacterType** | This class mapping between a SQL CHAR and a Java Character. |
| **CharArrayType** | It is a collection of VARCHAR like: char[]. |
| **CharBooleanType** | This is a superclass that can be mapping between SQL CHAR and Java boolean. |
| **ClassType** | This class mapping between SQL VARCHAR and Java class. |
| **ClobType** | This class mapping between SQL CLOB and java.sql.Clob. |
| **CollectionType** | It handles the Hibernate PersistentCollections. |
| **ComponentType** | This class mapping all components. |
| **CompositeCustomType** | It adjusts CompositeUserType to Type interface.; |
| **CurrencyType** | It is used to mapping a SQL VARCHAR into a java.util.Currency. |
| **CustomCollectionType** | This class is created by users and implement the PersistentCollection. |
| **CustomType** | It adjusts the user type to generic type interface and changes the internal type contracts. |
| **DateType** | This class mapping between an SQL DATE to Java Date. |
| **DbTimestampType** |  |
| **DoubleType** | It can be used to mapping between SQL DOUBLE to Java Double. |
| **EmbeddedComponentType** |  |
| **EntityType** | It communicates an entity class. |
| **FloatType** | It links SQL FLOAT to Java Float. |
| **ForeignKeyDirection** | This class shows directionality of the foreign key constraint. |
| **IdentifierBagType** |  |
| **ImmutableType** | This is a superclass of the nullable immutable type. |
| **IntegerType** | This class links between the SQL INT to Java Integer. |
| **ListType** |  |
| **LocaleType** | It links into an SQL VARCHAR and a Java Locale. |
| **LongType** | This class links into an SQL BIGINT and a Java Long. |
| **ManyToOneType** | It associates many-to-one entity. |
| **MapType** |  |
| **MetaType** |  |
| **MutableType** | This is a superclass of mutable nullable types. |
| **NullableType** | This is a supperclass of single-column nullable types. |
| **OneToOneType** | It associates one-to-one entity. |
| **OrderedMapType** |  |
| **OrderedSetType** |  |
| **PrimitiveType** | This is the superclass of primitive or primitive wrappers types. |
| **SerializableType** | This class maps an SQL VARBINARY to a serializable Java object. |
| **SetType** |  |
| **ShortType** | It communicates between an SQL SMALLINT and Java Short. |
| **SortedMapType** |  |
| **SortedSetType** |  |
| **SpacialOneToOneType** |  |
| **StringType** | It communicates between an SQL VARCHAR and Java String. |
| **TextType** | This class links between an SQL CLOB and Java String. |
| **TimestampType** | This class mapping between an SQL TIMESTAMP and Java java.util.Date or java.sql.Timestamp. |
| **TimeType** | It maps an SQL TIME to  Java java.util.Date or java.sql.Time. |
| **TimeZoneType** | It communicates an SQL  VARCHAR to Java java.util.TimeZone. |
| **TrueFalseType** | This class maps an SQL CHAR to a Java Boolean. |
| **TypeFactory** | This class used internally and holds an instance of Type. |
| **WrapperBinaryType** |  |
| **YesNoType** | It maps between an SQL CHAR to Java Boolean. |

|  |  |
| --- | --- |
| **Exceptions and Descriptions:** | |
| **SerializationException** | This exception occurs when the property could not be serialized or deserialized. |