Using DataDirect® JDBC™ Drivers with Hibernate

What is Hibernate?

Hibernate is an open source project that provides an object-relational mapping solution for Java applications. Business logic in Java applications uses objects to represent data, while databases store relational data in a table format consisting of rows and columns.

Hibernate maps the data represented in Java objects to the relational data of the database. Each row of a table is represented as a single Java object. Hibernate also supplies a persistence layer for saving, updating and retrieving data in your database tables, as well as connection pool and transaction management facilities.

Hibernate was created to address the needs of developers who wish to easily persist their Java objects using a robust persistence mechanism while avoiding the complexity of EJB 2.0 persistence. Hibernate provides applications an object relational data mapping solution while completely abstracting the power of the underlying data source.

Why Using DataDirect JDBC Drivers Makes This Technology Better

- Performance and Scalability
  - Best JDBC driver performance for any production scenario
  - Best JDBC performance for single-threaded (one connection) and multi-threaded (many connections) situations
- Quality and Support
  - Quality tested with extensive internal and industry test suites
  - Embedded in the world's most demanding software applications and application servers
  - Backed by the industry's best technical support organization with a complete focus on data access middleware
- Consistent and complete database support in one package
  - Robust database feature support including security features such as SSL and Kerberos
  - Supports the latest versions of all major databases
- Standards-based, 100% Java approach to feature implementation across databases.

DataDirect™ TECHNOLOGIES

Required Components

- Download Hibernate Core from http://www.hibernate.org/6.html
- Download DataDirect Connect *for* JDBC from http://www.datadirect.com/downloads/registration/connect_jdbc/index.ssp

Installation of DataDirect Connect *for* JDBC Drivers

After downloading the DataDirect Connect *for* JDBC drivers, they may be installed with a 15 day evaluation license. DataDirect Connect *for* JDBC supports Microsoft SQL Server, DB2, Oracle, Sybase, Informix, and MySQL. This example will use only the SQL Server driver. Complete documentation along with the DataDirect Connect *for* JDBC installation guide is available at http://www.datadirect.com/techres/jdbcproddoc/index.ssp

Required Hibernate and DataDirect Connect *for* JDBC Libraries

For simplicity, all Java source and configuration files, as well as the required Hibernate and DataDirect Connect *for* JDBC libraries, will be placed in the same location. This will be our examples working directory.

Copy the following required Hibernate libraries from the Hibernate installation to the working directory. Some of the file names in your Hibernate installation will include numbers indicating the version of that file. The version numbers have been omitted from these file names below. These libraries must be on your classpath when you run the example.

lib/antlr.jar

lib/cglib.jar

lib/asm.jar

lib/asm-attrs.jars

lib/commons-collections.jar

lib/commons-logging.jar

lib/jta.jar

lib/dom4j.jar

lib/log4j.jar

hibernate3.jar

Copy the following DataDirect Connect *for* JDBC libraries from the driver installation to the working directory. These libraries must be on your classpath when you run the example.

lib/sqlserver.jar

lib/base.jar

lib/util.jar

Implementing an Object Relational Solution with Hibernate and DataDirect Connect *for* JDBC Drivers

To implement a simple object relational solution, four components will be required in addition to the previously mentioned Hibernate and DataDirect Java libraries. Two are Hibernate configuration files. The remaining two are Java classes. One Java class is our persistent class and the other is the sample application that will persist this class to and retrieve it from the database. Complete implementations of these components are included and an explanation of each is provided below.

The persistent class, `Emp`, represents the database table EMP. This class uses standard JavaBean naming conventions for property getter and setter methods, as well as private visibility for the fields. This is a recommended design but is not required. Each instance of this class represents a single row in the database table and each field of this class represents a specific column in the database table.

```
package tables;

import java.util.Date;

public class Emp {
     private String firstName;
     private String lastName;
     private int empId;
     private Date hireDate;
     private float salary;
     private String dept;
     private int exempt;
     private String interests;

     public String getFirstName() {
            return firstName;
     }

     public void setFirstName(String firstName) {
            this.firstName = firstName;
     }

     public String getLastName() {
            return lastName;
     }

     public void setLastName(String lastName) {
            this.lastName = lastName;
     }

     public int getEmpId() {
            return empId;
     }

     private void setEmpId(int empId) {
            this.empId = empId;
     }

     public Date getHireDate() {
            return hireDate;
     }

     public void setHireDate(Date hireDate) {
            this.hireDate = hireDate;
     }
```

```
    public float getSalary() {
           return salary;
    }

    public void setSalary(float salary) {
           this.salary = salary;
    }

    public String getDept() {
           return dept;
    }

    public void setDept(String dept) {
           this.dept = dept;
    }

    public int getExempt() {
           return exempt;
    }

    public void setExempt(int exempt) {
           this.exempt = exempt;
    }

    public String getInterests() {
           return interests;
    }

    public void setInterests(String interests) {
           this.interests = interests;
    }

                        }
```

The Hibernate mapping file tells Hibernate how it will map a specific Java object to a specific database table. This file specifies the database column that each field of the persistent class will map to as well as the Hibernate data type for each of these mappings. This file uses the `<id>` tag to specify which field of the persistent class will be used as a unique identifier. Since the name of our persistent class will be `Emp`, we will save the mapping file as `Emp.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="Emp" table="EMP">
      <id name="empId" column="EMP_ID">
         <generator class="native"/>
      </id>
      <property name="firstName" type="string" column="FIRST_NAME"/>
      <property name="lastName" type="string" column="LAST_NAME"/>
      <property name="hireDate" type="date" column="HIRE_DATE"/>
      <property name="salary" type="float" column="SALARY"/>
      <property name="dept" type="string" column="DEPT"/>
      <property name="exempt" type="integer" column="EXEMPT"/>
      <property name="interests" type="string" column="INTERESTS"/>
   </class>

                        </hibernate-mapping>
```

The main Hibernate configuration file in this example is `hibernate.cfg.xml`. This file contains information for creating the Hibernate session factory. It is where we will specify the driver to be used, its connectivity information, the appropriate database dialect and what objects will be mapped.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

   <session-factory>
      <!-- Database connection settings -->
      <property name="connection.driver_class">
           com.ddtek.jdbc.sqlserver.SQLServerDriver
      </property>
      <property name="connection.url">
           jdbc:datadirect:sqlserver://nc-answline-
w2k:1435;databaseName=test
      </property>
      <property name="connection.username">test</property>
      <property name="connection.password">test</property>


      <!-- SQL dialect -->
      <property
name="dialect">org.hibernate.dialect.SQLServerDialect</property>

      <!-- Enable Hibernate's automatic session context management -->
      <property name="current_session_context_class">thread</property>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">true</property>

      <!-- Drop and re-create the database schema on startup -->
      <property name="hbm2ddl.auto">create</property>

      <mapping resource="Emp.hbm.xml"/>

   </session-factory>


                        </hibernate-configuration>
```

Our sample application is the `EmpManager` class which contains a `main()` method and is a standalone Java Application. However, these concepts could be implemented in any Java application such as a servlet, EJB or JSP. The sample application establishes a Hibernate session and creates unique instances of our persistent class. It also implements methods that use Hibernate to persist these objects to the database and to retrieve them from the database. Hibernate generates all the appropriate SQL to create and drop the table as well as statements to insert to, select from and update the database table. These statements are executed on the database via the DataDirect Connect *for* JDBC driver.

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

import java.util.Date;
import java.util.List;

public class EmpManager {

    public static void main(String[] args) {

        EmpManager mgr = new EmpManager();

        Configuration config = new Configuration().configure();
        SessionFactory sessionFactory = config.buildSessionFactory();
        Session session = sessionFactory.getCurrentSession();
        session.beginTransaction();
```

```
        //Persist several instances of the Java object Emp to the database
table EMP
        mgr.createAndStoreObjects(session, "Tyler", "Bennett", new
Date("1977/06/01"),
                                  (float) 32000.00, "D101", 1, "");
        mgr.createAndStoreObjects(session, "John", "Rappl", new
Date("1987/07/15"),
                                  (float) 47000.00, "D050", 1, "");
        mgr.createAndStoreObjects(session, "George", "Woltman", new
Date("1982/08/07"),
                                  (float) 53500.00, "D101", 1, "");
        mgr.createAndStoreObjects(session, "Adam", "Smith", new
Date("1988/01/15"),
                                  (float) 18000.00, "D202", 0, "");
        mgr.createAndStoreObjects(session, "David", "McClellan", new
Date("1982/07/27"),
                                  (float) 41500.00, "D101", 1, "");

        /*Retrieve the persisted objects from the database and display them
on the
          console.*/
        mgr.listObjects(session);

        //Commit the transaction
        session.getTransaction().commit();

        //Close the current session factory and unbind it from the thread.
        sessionFactory.close();
    }

    private void createAndStoreObjects(Session session,
                                       String firstName,
                                       String lastName,
                                       Date hireDate,
                                       float salary,
                                       String dept,
                                       int exempt,
                                       String interests) {

        Emp theEmp = new Emp();
        theEmp.setFirstName(firstName);
        theEmp.setLastName(lastName);
        theEmp.setHireDate(hireDate);
        theEmp.setSalary(salary);
        theEmp.setDept(dept);
        theEmp.setExempt(exempt);
        theEmp.setInterests(interests);

        session.save(theEmp);

    }

    private void listObjects(Session session) {

     //Use the currect session to retrive records as a Java List of objects.
     Query hibernateQuery = session.createQuery("from Emp");
        List emps = hibernateQuery.list();
        for (int i = 0; i < emps.size(); i++) {
            Emp theEmp = (Emp) emps.get(i);
            System.out.println("Emp Id: " + theEmp.getEmpId());
            System.out.println("  First Name: " + theEmp.getFirstName());
            System.out.println("  Last Name: " + theEmp.getLastName());
            System.out.println("  Hire Date: " + theEmp.getHireDate());
            System.out.println("  Salary: " + theEmp.getSalary());
            System.out.println("  Dept: " + theEmp.getDept());
            System.out.println("  Exempt: " + theEmp.getExempt());
            System.out.println("  Interests: " + theEmp.getInterests());
        }
        return;
    }


}
```

Running the Sample Application

After setting the classpath as defined above, run the sample application from the command line as follows:

java EmpManager

Below is a sample of the output that the application generates showing that the Java objects were persisted to the database and subsequently returned to the application.

```
Hibernate: insert into EMP (FIRST_NAME, LAST_NAME, HIRE_DATE, SALARY, DEPT,
EXEM
PT, INTERESTS) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into EMP (FIRST_NAME, LAST_NAME, HIRE_DATE, SALARY, DEPT,
EXEM
PT, INTERESTS) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into EMP (FIRST_NAME, LAST_NAME, HIRE_DATE, SALARY, DEPT,
EXEM
PT, INTERESTS) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into EMP (FIRST_NAME, LAST_NAME, HIRE_DATE, SALARY, DEPT,
EXEM
PT, INTERESTS) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into EMP (FIRST_NAME, LAST_NAME, HIRE_DATE, SALARY, DEPT,
EXEM
PT, INTERESTS) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: select emp0_.EMP_ID as EMP1_0_, emp0_.FIRST_NAME as FIRST2_0_,
emp0_.
LAST_NAME as LAST3_0_, emp0_.HIRE_DATE as HIRE4_0_, emp0_.SALARY as
SALARY0_, em
p0_.DEPT as DEPT0_, emp0_.EXEMPT as EXEMPT0_, emp0_.INTERESTS as
INTERESTS0_ fro
m EMP emp0_
Emp Id: 1
  First Name: Tyler
  Last Name: Bennett
  Hire Date: Wed Jun 01 00:00:00 EDT 1977
  Salary: 32000.0
  Dept: D101
  Exempt: 1
  Interests:
Emp Id: 2
  First Name: John
  Last Name: Rappl
  Hire Date: Wed Jul 15 00:00:00 EDT 1987
  Salary: 47000.0
  Dept: D050
  Exempt: 1
  Interests:
Emp Id: 3
  First Name: George
  Last Name: Woltman
  Hire Date: Sat Aug 07 00:00:00 EDT 1982
  Salary: 53500.0
  Dept: D101
  Exempt: 1
  Interests:
Emp Id: 4
  First Name: Adam
  Last Name: Smith
  Hire Date: Fri Jan 15 00:00:00 EST 1988
  Salary: 18000.0
  Dept: D202
  Exempt: 0
  Interests:
Emp Id: 5
  First Name: David
  Last Name: McClellan
  Hire Date: Tue Jul 27 00:00:00 EDT 1982
```

```
Salary: 41500.0
Dept: D101
Exempt: 1

                                Interests:
```

Useful Links

Hibernate documentation:

http://www.hibernate.org/5.html

Download Hibernate Core:

http://www.hibernate.org/6.html

Download DataDirect Connect *for* JDBC:

http://www.datadirect.com/downloads/registration/connect_jdbc/index.ssp

DataDirect Connect *for* JDBC documentation
http://www.datadirect.com/techres/jdbcproddoc/index.ssp

We welcome your feedback! Please send any comments concerning documentation, including suggestions for other topics that you would like to see, to:

docgroup@datadirect.com

FOR MORE INFORMATION

800-876-3101

Worldwide Sales

Belgium (French).....0800 12 045
Belgium (Dutch).....0800 12 046
France.....0800 911 454
Germany 0800 181 78 76
Japan 0120.20.9613
Netherlands.....0800 022 0524
United Kingdom.....0800 169 19 07
United States.....800 876 3101

DataDirect Technologies is the software industry's only comprehensive provider of software for connecting the world's most critical business applications to data and services, running on any platform, using proven and emerging standards. Developers worldwide depend on DataDirect® products to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC™ and ADO.NET, XQuery and SOAP. More than 300 leading independent software vendors and thousands of enterprises rely on DataDirect Technologies to simplify and streamline data connectivity for distributed systems and to reduce the complexity of mainframe integration. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS). For more information, visit www.datadirect.com.