**Author: AbdulKarim Egal (working with Yang Zheng)**

**Draw Backs of JDBC:**

* Handling the JDBC connections and properly closing the connection is also a big issue. Properly closing the connection is must.
* There will be several places in the java code that will SQL commands or Queries in JDBC.
* Any database changes will require java code changes. In JDBC, after the application has created if the table structure is modified then the application will not work therefore application developers will need to modify and compile and re-deploy required, which is time costly and tedious.

**Introduction to Hibernate**

Hibernate is a powerful and high-performance ORM (Object-relational Mapping) service. It simply means storing a Java object in a (relational) database. When stored, Hibernate automatically maps (places) and stores each instance variable value of a Java object in a database column.

ORM is a supplement to JDBC and not a replacement. Hibernate internally uses JDBC. The aim of ORM is to reduce the writing of number of lines of DB programming code in Java.

**For example:**

An Employee **object**emp has three properties (instance variables) like **emp.id, emp.name andemp.salary**. When the **object emp** is stored in the database, Hibernate stores these variables values in the columns **EMPID, EMPNAME** and E**MPSAL**of a database table. This reduces the code of JDBC to a maximum extent which a programmer is most accustomed earlier. Java variables vs table columns mapping is done in an XML file (like web.xml, incase of servlets, where an alias name is mapped to the actual servlet).

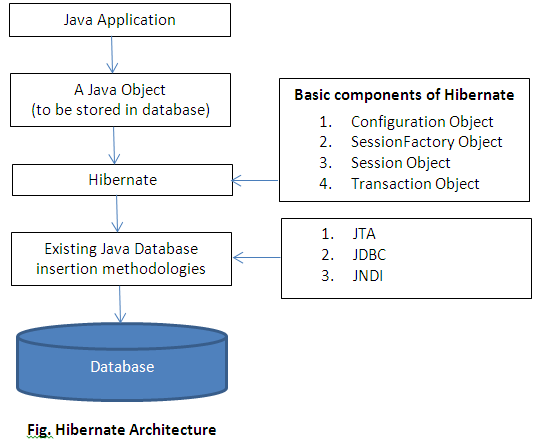
Hibernate is object/relational persistence for Java . Here, persistence means writing Java object to a permanent storage and making it to live (stay) long. Always persistence storage comes with writing to a hard disk like to a text file or a RDBMS table. Remember DS like an ArrayList or Vector gives a temporary storage (available to program until program terminates. Once the program execution is over, the data is lost and when you restart the same program, the data is not available) of data as the data is written to the RAM. Object/relational means storing or writing a Java object to a database (to be more precisely, relational database) like Oracle and mySQL. Storing the variables of an object is known as storing the state of the object or persisting the object.

Hibernate makes use of persistent objects commonly called as POJO (POJO = "Plain Old Java Object".) along with XML mapping documents for persisting objects to the database layer. The term POJO refers to a normal Java objects that does not serve any other special role or implement any special interfaces of any of the Java frameworks (EJB, JDBC, DAO, JDO, etc...).

Hibernate is an open source code that can be freely downloaded from www.hibernate.org and used. Hibernate comes with three major versions:

* Hibernate 1.0
* Hibernate 2.0
* Hibernate 3.0.

**Hibernate Architecture**

[](http://way2java.com/wp-content/uploads/2013/12/image118.png)

Hibernate uses JDBC internally to insert a Java object into database. The main objects of Hibernate used in coding are Configuration, SessionFactory, Session and Transaction. The insertion of an object in database is known as persisting (literal meaning: preserving) a Java object.

First step is hibernate application is to retrieve Hibernate Session; Hibernate Session is the main runtime interface between a Java application and Hibernate. SessionFactory allows applications to create hibernate session by reading hibernate configurations file hibernate.cfg.xml.  
  
After specifying transaction boundaries, application can make use of persistent java objects and use session for persisting to the databases.

**Typical Hibernate code**

|  |
| --- |
| sessionFactory = new Configuration().configure().buildSessionFactory();  Session session = sessionFactory.openSession();  Transaction tx = session.beginTransaction();  Customer newCustomer = new Customer(); newCustomer.setName("New Customer"); newCustomer.setAddress("Address of New Customer"); newCustomer.setEmailId("NewCustomer@NewCustomer.com");  session.save(newCustomer);  tx.commit();  session.close(); |

**Benefits of using Hibernate**

**Relational Persistence for JAVA**

Working with both Object-Oriented software and Relational Database is complicated task with JDBC because there is mismatch between how data is represented in objects versus relational database. So with JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. Hibernate is flexible and powerful ORM (Object-relational Mapping) solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

**Transparent Persistence**

The automatic mapping of Java objects with database tables and vice versa is called Transparent Persistence. It is also defined as writing Java Object to a Database and reading back. The term “Transparent” means unknown to the programmer, to say, the internal database implementation is unknown.

Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. With JDBC this conversion is to be taken care of by the developer manually with lines of code.

**Support for Query Language**

JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e to select effective query from a number of queries to perform same task. Hibernate provides a powerful query language Hibernate Query Language – HQL (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application.

**Database Dependent Code**

Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object to-table. Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties.

**Maintenance Cost**

With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost.

**Optimize Performance**

Caching is retention of data, usually in application to reduce disk access. Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code.

**Automatic Versioning and Time Stamping**

By database versioning one can be assured that the changes done by one person is not being roll backed by another one unintentionally. Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow to save it because this user does not has updated data. In JDBC there is no check that always every user has updated data. This check has to be added by the developer.

**Open-Source, Zero-Cost Product License**

Hibernate is an open source and free to use for both development and production deployments.

**Enterprise-Class Reliability and Scalability**

Hibernate scales well in any environment, no matter if use it in-house Intranet that serves hundreds of users or for mission-critical applications that serve hundreds of thousands. JDBC cannot be scaled easily.

**Disadvantages of Hibernate**

1. Steep learning curve.

Lots of API to learn: A lot of effort is required to learn Hibernate. So, not very easy to learn hibernate easily.

1. Use of Hibernate is an overhead for the applications which are:

* simple and use one database that never change
* need to put data to database tables, no further SQL queries
* there are no objects which are mapped to two different tables

Hibernate increases extra layers and complexity. So for these types of applications JDBC is the best choice.

1. Support for Hibernate on Internet is not sufficient.
2. Anybody wanting to maintain application using Hibernate will need to know Hibernate.

1. For complex data, mapping from Object-to-tables and vise-versa reduces performance and increases time of conversion.
2. Hibernate does not allow some type of queries which are supported by JDBC.

For example, Hibernate does not allow inserting multiple objects (persistent data) to same table using single query. Developer has to write separate query to insert each object.