Volume 2, No. 6, June 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARICLE
Available Online at www.jgrcs.info

HIBERNATE TECHNOLOGY FOR AN EFFICIENT BUSINESS APPLICATION EXTENSION

B.Vasavi*[1], Y.V.Sreevani[2], G.Sindhu Priya[3]

*[1] Associate Professor, Department of Computer Science and Engineering
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P, India.
vasavi.bande@yahoo.co.in
[2] Associate Professor, Department of Computer Science and Engineering
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P, India
s_vanikumar@yahoo.co.in
[3] Student, B.Tech Final Year, Department of Computer Science and Engineering
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P, India
sindhupriya78@gmail.com

Abstract :This paper discusses hibernate technology as a novel and efficient means to access huge databases and also focuses on how to implement persistent features in object-oriented system through it . It discusses currently available hibernate mapping framework in detail. Hibernate provides support for collections, object relations, as well as complex and composite types. In addition to persisting objects, hibernate also provides a rich query language to retrieve objects from the database, along with an efficient caching layer and Java Management Extensions (JMX) support. Hibernate is a powerful, high-performance, feature-rich and very popular ORM solution for Java. Hibernate facilitates development of persistent objects based on the common Java object model to mirror the underlying database structure. This approach progresses the business performance to some extent, advances development efficiency exceedingly and obtains preferable economical efficiency and practicability. In addition to, it compares and analyzes the database access efficiency resulted from two mechanisms based on Hibernate and JDBC. This paper offers insight into hibernate technology its implementation and usage.

Keywords: Hibernate, HQL, ORM,, Database, SQL.

INTRODUCTION

A major portion of the development of an enterprise application involves the creation and maintenance of the persistence layer used to accumulate and retrieve objects from the database of choice [1]. Many organizations resort to create homegrown, often buggy, persistence layers. If changes are made to the underlying database schema, it can be expensive to disseminate those changes to the rest of the application. Hibernate steps in to fill this gap, providing an easy-to-use and powerful object relational persistence framework for Java applications. Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves Object-Relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. ORM is a piece of software product for the representation and translation of data between the database and the object-oriented programming language. Hibernate is one such ORM solution and it is an open-source project. The Hibernate 2.1 framework has won a award in 2005. Hibernate provides support for collections and object relations, as well as composite types. It also provides a rich query language to retrieve objects from the database, a competent caching layer and has Java Management Extensions (JMX) support. Hibernate is released under the lesser GNU Public License, which is sufficient for use in commercial as well as open source applications. It supports numerous databases, including Oracle and DB2, also popular open source databases such as PostgreSQL and MySQL.

Working of Hibernate

Rather than utilizing bytecode processing or code generation, hibernate uses runtime reflection to determine the persistent properties of a class. The objects to be persisted are defined in a mapping document, which serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object. The mapping documents are compiled at application startup time and supply the framework with necessary information for a class [2]. In addition to it, they are used in support operations, such as generating the database schema or creating stub and Java source files. A Session Factory is created from the compiled collection of mapping documents[6]. The Session Factory provides the mechanism for managing persistent classes and the Session interface. The Session class provides the interface between the persistent data store and the application. The Session interface wraps a JDBC connection, which can be user-managed or controlled by hibernate, and is only intended to be used by a single application thread, then closed and discarded.

Hibernate Architecture

The following Figure1and Figure 2 describes the high level architecture of hibernate i.e. they show how hibernate uses the database and configuration data to provide persistence services (and persistent objects) to an application. To use Hibernate, it requires creating Java classes that represent 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. The Figure1describes the high level architecture of hibernate.

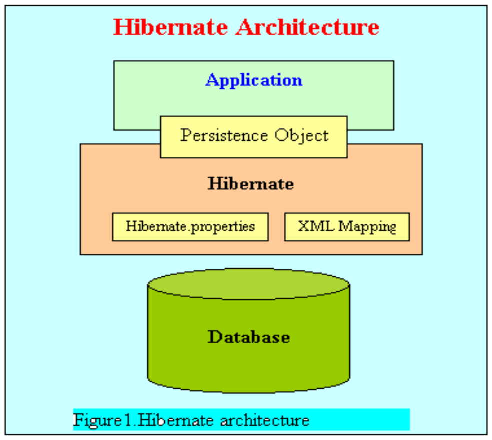

Figure 1. Basic Hibernate Architecture

Hibernate architecture has following three main components.

Connection management:

Hibernate Connection management service grant efficient management of the database connections. Database connection is the priciest part of database as it requires a lot of resources of open/close the database connection.

Transaction management**:**

Transaction management service provides the capability to the user to execute more than one database statements at a time.

Object relational mapping:

Object relational mapping is a 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, view 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 extremely good tool as far as object relational mapping is concern, but in terms of connection management and transaction management, it lacks 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 usage. It is called "Lite" architecture when we only use object relational mapping component. While in "Full Cream" architecture all the three component Object Relational mapping, Connection Management and Transaction Management are used. Hibernate architecture can be shown in detail in the figure 2.

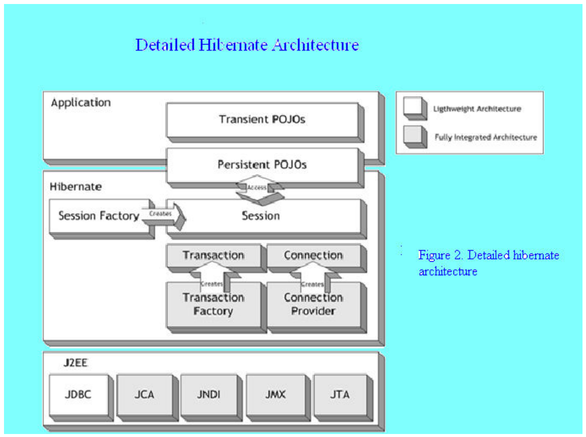

Figure 2. A detailed Hibernate Architecture

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 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 [4]. 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 and is less verbose than SQL. There are other options that can be used while using hibernate [3]. These are **Query By Criteria (QBC)** and **Query BY Example (QBE) -**using Criteria API and the **Native SQL** queries.

Full support for relational operations- HQL permits 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 eliminates the need of creating the object and populates the data from result set.

Polymorphic Queries- HQL fully supports polymorphic queries. Polymorphic queries provide 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 advanced features such as pagination, fetch and join with dynamic profiling, Inner/outer/full joins and 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.

UNDERSTANDING AND IMPLEMENTATION OF HIBERNATE MAPPING

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping SYSTEM
"Hibernate_Mapping.dtd">
<hibernate-mapping>
 <class name="sample" table="COMMUNICATE">
<idname="id"type="long"column="ID"><generator
class="assigned"/> </id>
<property name="firstName"> <column
name="FIRSTNAME"/> </property>
<property name="lastName"> <column
name="LASTNAME"/> </property>
 </class>
 </hibernate-mapping>
```

Hibernate mapping documents are simple xml documents [10].

Here are some 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 [3].

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 is 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 COMMUNICATE. The attributes of the id element are:

   a) **Name:** The property name used by the persistent class.

   b) **Column:** The column used to store the primary key value.

   c) **Type:** The Java data type is used.

   d) **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**>** method is used to generate the primary key for the new record. Here is some of the frequently used generators.

   a) **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.

   b) **Sequence** - Hibernate can also use the sequences to generate the primary key. It can be used with DB2, postgreSQL , Oracle, SAPDB databases.

   c) **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.

Configuring Hibernate

Hibernate can be configured by creating a property file named hibernate properties in the *src* directory and adding its path to the application's classpath. This file consists of the properties used by Hibernate to connect to database[11], generate schema, and obtain other database-specific information. To reflect changes in the underlying database into the whole application, only values of the properties in this file need to be modified. Model 1 shows a simple example. Most of these properties are self-explanatory. To set up MySQL Database in the configuration file i.e. hibernate.cfg.xml.Considering database running on the localhost. So, create the database ("hibernate tutorial") on the MySQL server running on local host.

Model 1: A simple example of hibernate properties file:

```
hibernate.connection.driver_class =
COM.ibm.db2.jdbc.app.DB2Driver
hibernate.connection.url = jdbc: db2: inventory
hibernate.connection.username = db2admin
hibernate.connection.password = Taman
hibernate.dialect = cirrus.hibernate.sql.DB2Dialect
```

Hibernate also can be configured by using a simple XML file named hibernate.cfg.xml, which exists inside the *src* directory. This file's structure is very similar to hibernate properties and has the same functionality. This shown as example
in the following model.

Model 2: Example of a simple hibernate.cfg.xml

```
<? xml version="1.0"?>
    <!DOCTYPE hibernate-mapping SYSTEM
"Hibernate_Mapping.dtd">
```

```
<hibernate-mapping>
<calss name="sample" table="COMMUNICATE">
<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>
</class>
 </hibernate-mapping>
```

Building an Application with Hibernate

We can build a application by adhering to the following steps.

- Creating mapping documents.
- Generating stub Java classes for persistent objects.
- Generating database schema.
- Preparing code to initialize and run Hibernate in an appropriate place .

These steps are explained in the following sections.

Creating Mapping Documents

Mapping documents are XML documents used to define the persistent objects and contain information about an object's persistent fields, associations, subclasses, and proxies, if any [3]. One mapping document is created for each persistent object and saved in a file with the name class_name.hbm.xml, where class_name is the name of the object's class. Model 3 and Model 4 gives an example of mapping document Event.hbm.xml. The mapping documents are compiled at application start-up to provide hibernate with information about the persistent objects' corresponding classes, their respective structures, to which database table should they be mapped, and how. Hibernate also uses these mapping documents to generate corresponding database schema and stub Java classes for the persistence layer, using inbuilt utilities called *SchemaExport* and *CodeGenerator* respectively.

Model 3: An example of mapping document Event.hbm.xml

```
<! DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 2.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
 <Hibernate-mapping>
 <class name="com.myPackage.myApplication.Event" table="EVENTS">
 <id name="id" column="uid" type="long">
 <generator class="increment"/>
 </id>
 <property name="date" column="event_date" type="timestamp"/>
<property name="title" column="event_title" type="string"/>
</class>
</hibernate-mapping>
```

Generating Stub Classes

This task becomes simpler after mapping documents are created. Stub classes can be created by using Hibernate's built-in utility CodeGenerator by executing a simple command. Command's syntax is given below:

```
java -cp classpath net.sf.hibernate.tool.hbm2java.CodeGenerator: options mapping files.
```

Provide appropriate values for the class path, options, and mapping files parameters. Model 4 shows the stub file generated using the mapping document given in model 3.

Model 4: Stub Classes

```
package com.myPackage.myApplication;
public class Event {
   private String title;
   private Date date;
   private Long id;
   Event(){ }
   public Long getId() {
      return id;
   }
   private void setId(Long id) {
      this.id = id;
   }
   public Date getDate() {
      return date;
   }
   public void setDate(Date date) {
      this.date = date;
   }
   public String getTitle() {
      return title;
   }
   public void setTitle(String title) {
      this.title = title;
   }
}
```

Generating Database Schema

Mapping files in hand, it's time to generate the database schema [7]. Hibernate ships with the SchemaExport utility that will create the schema necessary for the mapping documents. This utility may be run from the command line or from an build script to connect to the database and create the schema, or to export the schema to a file. To generate database schema using Hibernate's SchemaExport, execute the following command after substituting appropriate values for parameters:

java -cp *classpath* net.sf.hibernate.tool.hbm2ddl.SchemaExport *options mapping_files.*

Provide appropriate values for the *classpath*, *options*, and *mapping_files* parameters. Figure 3 shows the schema generated using the mapping document given in model 3.

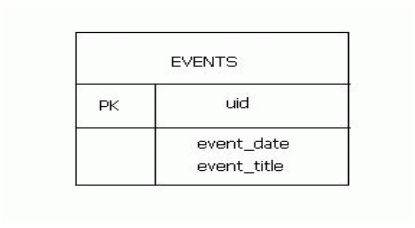

Figure 3: Graphical representation of schema as per Model 3

Initializing and Running Hibernate

To initialize and run hibernate, the following steps are to be taken [8]:

- Inside an appropriate class, instantiate and populate the desired object to be persisted.
- Obtain the net.sf.hibernate.SessionFactory object using the net.sf.hibernate.cfg.Configuration object at the start of the application.
- Open net.sf.hibernate.Session by calling the openSession() method on the SessionFactory object.
- Save the desired object and close the Session.

Model 5 shows how to implement the steps described above using a simple class. Now, the application is complete and, when executed, saves the desired objects to the underlying database and makes them persistent.

Model 5: A class for initializing Hibernate and making objects persistent.

```
package com.myPackage.myApplication;
import net.sf.hibernate.SessionFactory;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Transaction;
import net.sf.hibernate.cfg.Configuration;
public class EventManager {
public static void main(String[] args) {
      // Instantiate and populate object to be persisted
   Event ev = new Event();
   ev.setDate("22/6/2011")
   ev.setTitle("Hibernate startup ");
   try {
   //Start Hibernate
   Configuration cfg = new
   Configuration().addClass(Event.class);
   SessionFactory sf = cfg.buildSessionFactory();
   //Open Session
   Session sess = sf.openSession();
   } catch (HibernateException e) {
     e.printStackTrace();
   }
      //Save Product and close Session
      Transaction t = sess.beginTransaction();
      sess.save(ev);
      t.commit();
      sess.close();

   }

}
```

CREATING THE SESSION FACTORY

The SessionFactory stores the compiled mapping documents specified when the factory is created. Configuring the SessionFactory is fairly straightforward as shown in figure 4. All of the mappings are added to an instance of net.sf.hibernate.cfg.Configuration, which is then used to create the SessionFactory instance.

```
Configuration cfg = new Configuration()
addClass(example.Player.class)
addClass(example.Team.class);
SessionFactory factory = cfg.buildSessionFactory();
```

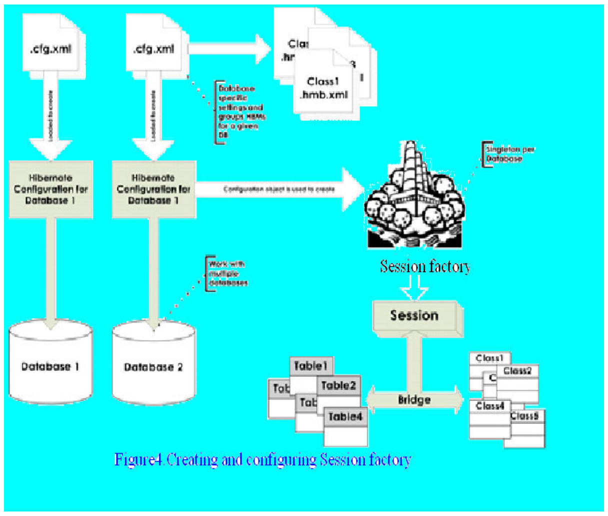
.cfg.xml
.cfg.xml
Class1
.hmb.xml
Database specific settings and groups HBMs for a given DB
Loaded to create
Loaded to create
Hibernate Configuration for Database 1
Hibernate Configuration for Database 1
Configuration object is used to create
Singleton per Database
Session factory
Work with multiple databases
Session
Database 1
Database 2
Table1
Table2
Table4
Bridge
Class1
Class2
Class4
Class5
Figure4.Creating and configuring Session factory

Figure 4: Session Factory Mapping

The Configuration class is only needed for the creation of the SessionFactory and can be discarded after the factory is built. Instances of Session are obtained by calling SessionFactory.openSession(). The logical lifecycle of a Session instance is the span of a database transaction. The SessionFactory can also be configured using an XML mapping file, placed in the root of your classpath [8]. The evident advantage to this approach is that your configuration isn't hardcoded in the application.

INTERCEPTORS IN HIBERNATE

Hibernate provides an ORM solution for persisting and querying data in the database. A Hibernate application can be structured in a way such that certain methods can be made and to be invoked when a particular life-cycle event occurs. Not always the API in a software/product will completely satisfy the application needs and requirements. Hibernate is no more away from this [9]. Therefore, Hibernate API is designed in such a way to provide pluggable framework through the concept of Interceptors. In a multi-tiered application, the situation for the inclusion of Interceptors can happen at any level. It can happen at the Client level, Server level and even at the persistence level. Imagine an application is saving employee records in a database and now the application mandates to display to the Database admin about the history of inserts and updates.

A simple general overview of the logic looks like the following,
- Insert/Update the records in the Database
- During Insert/Update, maintain the log information in a file as we can see, the maintenance of this logging information should happen whenever an insert/update goes to the Database. Such a logger interceptor can be easily plugged into the application with minimal code change because of the flexible design of hibernate.

Types of Interceptors

Based on their scope, Interceptors in hibernate can fall under two categories. They are,

- Application-scoped Interceptors
- Session-scoped Interceptors

Application-scoped Interceptor

An application can contain one or more database sessions represented by the Session interface. If an application is configured to use Global Interceptors, then it will affect the persistent objects in all the sessions. The following code configures a global interceptor.

```
SessionFactory sessionFactory =
configuration.buildSessionFactory();
Session session1 = sessionFactory.openSession();
Employee e1, e2 = null;
// Assume e1 and e2 objects are associated with session1.
Session session2 = sessionFactory.openSession();
User u1, u2 = null
//Assume u1 and u2 objects are associated with session1.
```

A global-scoped interceptor can be set to an application by calling the Configuration.setInterceptor(Interceptor) method. In the above code, we have two different session objects 'session1' and 'session2'. Let us assume that e1 and e2 Employee objects are associated with session 'session1' and u1 and u2 are the user objects associated with session 'session2'. The applied application-scoped interceptor would have affected all the objects (e1, e2, u1 and u2), even though they are in different sessions.

Session-*scoped Interceptor*

A session-scoped interceptor will affect all the persistent objects that are associated with that particular session only. The following code shows how to configure a session-scoped interceptor.

```
Configuration configuration = new Configuration();
SessionFactory sessionFactory =
configuration.buildSessionFactory();
MyInterceptor myInterceptor = new MyInterceptor();
Session session1 =
sessionFactory.openSession(myInterceptor);
Employee e1, e2 = null;
// Assume e1 and e2 objects are associated with session
'session1'.
MyAnotherInterceptor myAnotherInterceptor = new
MyAnotherInterceptor ();
Session session2 =
sessionFactory.openSession(myAnotherInterceptor);
User u1, u2 = null;
// assume u1 and u2 objects are associated with session
'session2'.
```

From the above code, we can infer that a session-scoped interceptor can be set by calling the method SessionFactory.openSession(Interceptor). In the above code, we have two different session objects 'session1' and 'session2' being configured with Interceptors MyInterceptor and MyAnotherInterceptor respectively. So, e1 and e2 objects will be affected by MyInterceptor, whereas u1 and u2 objects will be affected by MyAnotherInterceptor.

IMPLEMENTATION FOR TESTING HIBERNATE APPLICATION

Now we are ready to write a program to insert the data into database. We should first understand about the hibernates Session. Hibernate Session is the main runtime interface between a Java application and hibernate [7]. First we are required to get the hibernate Session. SessionFactory allows application to create the hibernate Session 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(communicate)

Here is the code of First Example1.java

```
package roseindia.tutorial.hibernate;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
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 Communicate and set
values in it by reading them from form object
System.out.println("Inserting Record");
Communicate contact = new Communicate();
contact.setId(3);
contact.setFirstName("jasmine");
contact.setLastName("Rao");
contact.setEmail("jasmine_74@yahoo.com");
session.save(contact);
System.out.println("Done");
```

Retrieving Persistent Classes

If you know the primary key value of the object that you want to retrieve, you can load it with the Session.load() method[6]. This method is overloaded to offer support for standard classes and BMP entity beans. To retrieve a persistent class without knowing its primary key value, you can use the Session.find() methods. The find () method allows you to pass an HQL (Hibernate Query Language) statement and retrieve matching objects as a java.util.List. The find () method has three signatures, allowing you to pass arguments to JDBC-like "?" parameters as a single argument, named parameters, or as an Object[].

Deleting Persistent Classes

Making a persistent object transient is accomplished with the Session.delete () method. This method supports passing either a specific object to delete or a query string to delete multiple objects from the database.

```
// method 1 – deleting the Player.
session.delete(player);
// Example  2 – deleting all of the Players with a salary greater than 4 million
session.delete("from player in class example.Player where player.annualSalary > 4000000");
```

It's important to note that while the object may be deleted from the database, your application may still hold a reference to the object. Deleting an object with collections of objects, such as the Team's set of Players, can cascade to child objects by specifying cascade="delete" for the set element in the mapping document.

Collections

Hibernate can manage the persistence of object collections [5], whether they are Sets, Maps, Lists, arrays of objects or primitive values. It also allows a different form of collection called a "bag". A bag can be mapped to a Collection or List, and contains an unordered, unindexed collection of entities [6]. Bags can contain the same element many times. Additional semantics supported by implementing classes, such as Linked List, are not maintained when persisted. Another note is that the property of a collection must be the interface type (List, Map, Set). This is because, in order to support lazy collections, hibernate uses its own implementations of the List, Map or Set interfaces. When accessing a lazily initialized collection, it's important to remember that a Session must be open, or an exception will be thrown as given below.

```
Session session = factory.openSession();
Team team = (Team) session.find("from team in class example.Team where
team.city = ?", cityName, Hibernate.STRING).get(0);
Set players = team.getPlayers();
session.close();
Player p = (Player) players.get(0); // exception will be thrown here
```

The exception is thrown because the Session needed to populate players was closed prematurely. Because of the potential for this bug, hibernate defaults to non lazy collections. However, lazy collections should be used for performance reasons.

Performance Considerations

Fortunately this functionality doesn't come at much of a performance cost. The hibernate website claims that its "overhead is much less than 10% of the JDBC calls," and our experience in deploying applications using hibernate supports this. Hibernate can make multiple optimizations when interacting with the database, including caching objects, efficient outer join fetching and executing SQL statements only when needed [9]. It is difficult to achieve this level of sophistication with hand-coded JDBC.

ADVANTAGES AND DISADVANTAGES OF HIBERNATE

Hibernate is better than plain JDBC**:** You can use hibernate which generates the SQL very easily and then automatically executes the necessary SQL statements [10]. This saves a lot of development and debugging time of the developer. Writing JDBC statement, setting the parameters, executing query and processing the result by hand is plenty of work. Hibernate will save all tiresome efforts.

Mapping of Domain object to relational database**:** Hibernate maps your domain object with the relational database [3]. Now you can concentrate on your business logic rather than managing the data in the database.

Light weight database**-**independent ORM solution

Layered architecture**:** Hibernate is layers architecture and you can use the components as per your application need.

JPA Provider**:** Hibernate can work as JPA provider in JPA based applications.

Standard ORM**:** Hibernate is standard ORM solutions and it also supports JPA.

Database Independent**:** Hibernate is database independent and you can use any database of your choice.

Caching Framework: There are many caching framework that works with Hibernate. You can use any one in your application to improve the performance of your application.

Disadvantages of Hibernate

Lots of API to learn**:** A lot of effort is required to learn Hibernate. Not so easy to learn hibernate easily

Debugging**:** Sometimes debugging and performance tuning becomes difficult.

Slower than JDBC**:** Hibernate is slower than pure JDBC as it is generating lots of SQL statements in runtime

Not suitable for Batch processing**:** It sensible to use pure JDBC for batch processing.

CONCLUSION

This paper has illustrated an introduction to what hibernate can do. The analyses how hibernate delivers a high performance, open source persistence framework comparable to many of its open source and commercial counterparts. Developers utilizing Hibernate can greatly reduce the amount of time and effort needed to code, test, and deploy applications. Hibernate is a powerful, high-performance, feature-rich and very popular ORM solution for Java along with mapping objects to a database. As discussed above hibernate also provides advanced data query and retrieval services through HQL, efficient caching, and other optimization techniques with useful built-in utilities for coding and schema generation. This automats the generation of a persistent layer to a large extent and hence, helps in relieving the developer up to 95% of common persistence related coding.

REFERENCES:

[1] Beginning Hibernate: from novice to professional, jefflinwood .

[2] http://www.devarticles.com.

[3] http://www.mindfiresolutions.com.

[4] Professional Hibernate (programmer to programmer),Ericpugh .

[5] http://www.apress.com.

[6] Java Persistence with Hibernate *Second Edition of Hibernate in Action* Christian Bauer and Gavin King.

[7] http://www.yangdaoqi.info.

[8] Hibernate in Action, Christian Bauer and Gavin King.

[9] Spring Persistence with Hibernate, AhmadSeddighi.

[10] Web development with java:using Hibernate,jsps and servlets,TimDowney.

[11] JBoss as 5 developments, Francescomarchoni.

Authors:

1. Vasavi Bande is M.Tech in Computer Science from Jawaharlal Nehru Technological University, A.P., India. She has vast experience in Computer Science and Engineering area pertaining to academics and industry related real time projects. She is presently working as Associate Professor in Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad [HITAM], A.P, India**.** She has presented many papers to her credit at National and National conferences. She has presided over as judge to many Paper Presentations and Technical Quizzes. She has authored 4 research papers and are published in reputed and indexed International Computer Science Journals. She has guided 20 Students of Master degree in Computer Science and Engineering in their major projects. She is bestowed with the Editorial Member on five International Journals Boards and is nominated as Reviewer to three International Journals. Her area of research includes TIBCO; Cloud computing, Network Security, Image Processing, Data Mining, Web Technologies and Emerging Technologies. She can be reached at: **vasavi.bande@yahoo.co.in**

2. Y.V.Sreevani Graduated in AM.I.E.T.E. from I.E.T.E, New Delhi, India, in 1997 and M.Tech in Computer science from Osmania University, Hyderabad, A.P., India in 2003. She has published 3 International papers. She is presently working as Associate Professor in Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad [HITAM], A.P, India**.** Her area of research includes Network Security, Data Mining, Web Mining Technologies and Emerging Technologies. She can be reached at: s_vanikumar@yahoo.co.in

3. G. Sindhu Priya is pursuing B.Tech in Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad [HITAM], A.P, India**.** She has participated in number of paper presentations and technical workshops. She is a noted and active member of the Sahaya Society, a organization to educate the less privileged school students i.e. www.sahayasociety.org. She has organized number of events which include Cultural, Symposiums and blood donation camps. She can be reached: sindhupriya78@gmail.com