**What is Hibernate – Hibernate Introduction**

## ****Draw Backs of JDBC:****

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* Here as a programmer we must close the connection, or we may get a chance to get out of connections message…!
* Actually if we didn’t close the connection in the finally block, then JDBC doesn’t responsible to close that connection.
* In JDBC we need to write SQL commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text.  Because JDBC doesn’t transfer objects directly.

In order to overcome above problems, Hibernate came into picture..!

## What is Hibernate?

Hibernate is the ORM tool given to transfer the data between a java (object) application and a database (Relational) in the form of the objects.  Hibernate is the open source light weight tool given by **Gavin King**, actually JBoss server is also created by this person only.

Hibernate is a non-invasive framework,  means it won’t forces the programmers to extend/implement any class/interface, and in hibernate we have all POJO classes so its light weight.

Hibernate can runs with in or without server, i mean it will suitable for all types of java applications (stand alone or desktop or any servlets etc.)

Hibernate is purely for persistence (to store/retrieve data from Database).

Or

Hibernate is a high-performance Object/Relational persistence and query service which is licensed under the open source GNU Lesser General Public License (LGPL) and is free to download. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities.

# Main Advantage And Disadvantages Of Hibernates

## Advantages of hibernates:

* Hibernate supports Inheritance, Associations, Collections
* In hibernate if we save the derived class object,  then its base class object will also be stored into the database, it means hibernate supporting inheritance
* Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List, Set, Map (Only new collections)
* In JDBC all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws.  Actually in hibernate we have the translator which converts checked to Un-checked ;)
* Hibernate has capability to generate primary keys automatically while we are storing the records into database
* Hibernate has its own query language, i.e. hibernate query language which is database independent
* So if we change the database, then also our application will works as HQL is database independent
* HQL contains database independent commands
* While we are inserting any record, if we don’t have any particular table in the database, JDBC will rises an error like “View not exist”, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us ;)f
* Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
* Hibernate supports annotations, apart from XML
* Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
* Getting pagination in hibernate is quite simple.

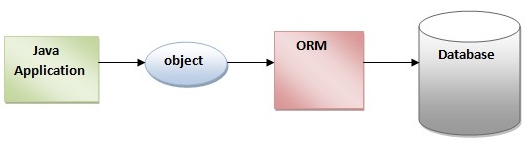
## Disadvantages of hibernates:

* I don’t think there are disadvantages in hibernate
* You know some thing.., Its saying hibernate is little slower than pure JDBC, actually the reason being hibernate used to generate many SQL statements in run time, but i guess this is not the disadvantage :-)
* But there is one major disadvantage, which was boilerplate code issue, actually we need to write same code in several files in the same application, but spring eliminated this

### Hibernate Framework

Hibernate framework simplifies the development of java application to interact with the database. Hibernate is an open source, lightweight, [ORM (Object Relational Mapping)](http://en.wikipedia.org/wiki/Object-relational_mapping) tool.

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

### Advantages of Hibernate Framework

There are many advantages of Hibernate Framework. They are as follows:

**1) Opensource and Lightweight:** Hibernate framework is open source under the LGPL license and lightweight.

**2) Fast performance:** The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

**3) Database Independent query:** HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, If database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

**4) Automatic table creation:** Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

**5) Simplifies complex join:** To fetch data form multiple tables is easy in hibernate framework.

**6) Provides query statistics and database status:** Hibernate supports Query cache and provide statistics about query and database status.

# Hibernate - ORM Overview

## What is JDBC?

JDBC stands for **Java Database Connectivity** and provides a set of Java API for accessing the relational databases from Java program. These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.

JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS without any modification.

## Pros and Cons of JDBC

|  |  |
| --- | --- |
| **Pros of JDBC** | **Cons of JDBC** |
| * Clean and simple SQL processing * Good performance with large data * Very good for small applications * Simple syntax so easy to learn | * Complex if it is used in large projects * Large programming overhead * No encapsulation * Hard to implement MVC concept * Query is DBMS specific |

## Why Object Relational Mapping (ORM)?

When we work with an object-oriented system, there's a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects. Consider the following Java Class with proper constructors and associated public function:

public class Employee {

private int id;

private String first\_name;

private String last\_name;

private int salary;

public Employee() {}

public Employee(String fname, String lname, int salary) {

this.first\_name = fname;

this.last\_name = lname;

this.salary = salary;

}

public int getId() {

return id;

}

public String getFirstName() {

return first\_name;

}

public String getLastName() {

return last\_name;

}

public int getSalary() {

return salary;

}

}

Consider above objects need to be stored and retrieved into the following RDBMS table:

create table EMPLOYEE (

id INT NOT NULL auto\_increment,

first\_name VARCHAR(20) default NULL,

last\_name VARCHAR(20) default NULL,

salary INT default NULL,

PRIMARY KEY (id)

);

First problem, what if we need to modify the design of our database after having developed few pages or our application? Second, Loading and storing objects in a relational database exposes us to the following five mismatch problems.

|  |  |
| --- | --- |
| **Mismatch** | **Description** |
| Granularity | Sometimes you will have an object model which has more classes than the number of corresponding tables in the database. |
| Inheritance | RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages. |
| Identity | A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)). |
| Associations | Object-oriented languages represent associations using object references whereas am RDBMS represents an association as a foreign key column. |
| Navigation | The ways you access objects in Java and in a RDBMS are fundamentally different. |

The **O**bject-**R**elational **M**apping (ORM) is the solution to handle all the above impedance mismatches.

## What is ORM?

ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc. An ORM system has following advantages over plain JDBC

|  |  |
| --- | --- |
| **S.N.** | **Advantages** |
| 1 | Let’s business code access objects rather than DB tables. |
| 2 | Hides details of SQL queries from OO logic. |
| 3 | Based on JDBC 'under the hood' |
| 4 | No need to deal with the database implementation. |
| 5 | Entities based on business concepts rather than database structure. |
| 6 | Transaction management and automatic key generation. |
| 7 | Fast development of application. |

An ORM solution consists of the following four entities:

|  |  |
| --- | --- |
| **S.N.** | **Solutions** |
| 1 | An API to perform basic CRUD operations on objects of persistent classes. |
| 2 | A language or API to specify queries that refers to classes and properties of classes. |
| 3 | A configurable facility for specifying mapping metadata. |
| 4 | A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions. |

## Java ORM Frameworks:

There are several persistent frameworks and ORM options in Java. A persistent framework is an ORM service that stores and retrieves objects into a relational database.

* Enterprise JavaBeans Entity Beans
* Java Data Objects
* Castor
* TopLink
* Spring DAO
* Hibernate
* And many more

# Hibernate - Architecture

The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.



Following is a detailed view of the Hibernate Application Architecture with few important core classes.



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API (JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

## Configuration Object:

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

* **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
* **Class Mapping Setup**

This component creates the connection between the Java classes and database tables.

## SessionFactory Object:

Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

## Session Object:

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

## Transaction Object:

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

## Query Object:

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

## Criteria Object:

Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

# Steps to create first Hibernate Application without IDE

1. [Steps to create first Hibernate Application](http://www.javatpoint.com/steps-to-create-first-hibernate-application)
   1. [Create the Persistent class](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step1)
   2. [Create the mapping file for Persistent class](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step2)
   3. [Create the Configuration file](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step3)
   4. [Create the class that retrieves or stores the persistent object](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step4)
   5. [Load the jar file](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step5)
   6. [Run the first hibernate application without IDE](http://www.javatpoint.com/steps-to-create-first-hibernate-application#step6)

Here, we are going to create the first hibernate application without IDE. For creating the first hibernate application, we need to follow following steps:

1. Create the Persistent class
2. Create the mapping file for Persistent class
3. Create the Configuration file
4. Create the class that retrieves or stores the persistent object
5. Load the jar file
6. Run the first hibernate application without IDE

### 1) Create the Persistent class

A simple Persistent class should follow some rules:

* **A no-arg constructor:** It is recommended that you have a default constructor at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.
* **Provide an identifier property (optional):** It is mapped to the primary key column of the database.
* **Declare getter and setter methods (optional):** The Hibernate recognizes the method by getter and setter method names by default.
* **Prefer non-final class:** Hibernate uses the concept of proxies, that depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

Let's create the simple Persistent class:

#### Employee.java

1. **package** com.javatpoint.mypackage;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String firstName,lastName;
7. **public** **int** getId() {
8. **return** id;
9. }
10. **public** **void** setId(**int** id) {
11. **this**.id = id;
12. }
13. **public** String getFirstName() {
14. **return** firstName;
15. }
16. **public** **void** setFirstName(String firstName) {
17. **this**.firstName = firstName;
18. }
19. **public** String getLastName() {
20. **return** lastName;
21. }
22. **public** **void** setLastName(String lastName) {
23. **this**.lastName = lastName;
24. }

27. }

### 2) Create the mapping file for Persistent class

The mapping file name conventionally, should be class\_name.hbm.xml. There are many elements of the mapping file.

* **hibernate-mapping** is the root element in the mapping file.
* **class** It is the sub-element of the hibernate-mapping element. It specifies the Persistent class.
* **id**It is the subelement of class. It specifies the primary key attribute in the class.
* **generator** It is the subelement of id. It is used to generate the primary key. There are many generator classes such as assigned (It is used if id is specified by the user), increment, hilo, sequence, native etc. We will learn all the generator classes later.
* **property** It is the subelement of class that specifies the property name of the Persistent class.

Let's see the mapping file for the Employee class:

#### employee.hbm.xml

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3. "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
6. <hibernate-mapping>
7. <**class** name="com.javatpoint.mypackage.Employee" table="emp1000">
8. <id name="id">
9. <generator **class**="assigned"></generator>
10. </id>
12. <property name="firstName"></property>
13. <property name="lastName"></property>
15. </**class**>
17. </hibernate-mapping>

### 3) Create the Configuration file

The configuration file contains informations about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml .

#### hibernate.cfg.xml

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3. "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
6. <hibernate-configuration>
8. <session-factory>
9. <property name="hbm2ddl.auto">update</property>
10. <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
11. <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
12. <property name="connection.username">system</property>
13. <property name="connection.password">oracle</property>
14. <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>
15. <mapping resource="employee.hbm.xml"/>
16. </session-factory>
18. </hibernate-configuration>

### 4) Create the class that retrieves or stores the object

In this class, we are simply storing the employee object to the database.

1. **package** com.javatpoint.mypackage;
3. **import** org.hibernate.Session;
4. **import** org.hibernate.SessionFactory;
5. **import** org.hibernate.Transaction;
6. **import** org.hibernate.cfg.Configuration;
8. **public** **class** StoreData {
9. **public** **static** **void** main(String[] args) {
11. //creating configuration object
12. Configuration cfg=**new** Configuration();
13. cfg.configure("hibernate.cfg.xml");//populates the data of the configuration file
15. //creating seession factory object
16. SessionFactory factory=cfg.buildSessionFactory();
18. //creating session object
19. Session session=factory.openSession();
21. //creating transaction object
22. Transaction t=session.beginTransaction();
24. Employee e1=**new** Employee();
25. e1.setId(115);
26. e1.setFirstName("sonoo");
27. e1.setLastName("jaiswal");
29. session.persist(e1);//persisting the object
31. t.commit();//transaction is commited
32. session.close();
34. System.out.println("successfully saved");
36. }
37. }

# Hibernate Query Language (HQL)

1. [Hibernate Query Language](http://www.javatpoint.com/hql)
2. [Advantage of HQL](http://www.javatpoint.com/hql)
3. [Query Interface](http://www.javatpoint.com/hql)

Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

### Advantage of HQL

There are many advantages of HQL. They are as follows:

* database independent
* supports polymorphic queries
* easy to learn for Java Programmer

### Query Interface

It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.

The query interface provides many methods. There is given commonly used methods:

1. **public int executeUpdate()** is used to execute the update or delete query.
2. **public List list()** returns the result of the ralation as a list.
3. **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
4. **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
5. **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
6. **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

### Example of HQL to get all the records

1. Query query=session.createQuery("from Emp");//here persistent class name is Emp
2. List list=query.list();

### Example of HQL to get records with pagination

1. Query query=session.createQuery("from Emp");
2. query.setFirstResult(5);
3. query.setMaxResult(10);
4. List list=query.list();//will return the records from 5 to 10th number

### Example of HQL update query

1. Transaction tx=session.beginTransaction();
2. Query q=session.createQuery("update User set name=:n where id=:i");
3. q.setParameter("n","Udit Kumar");
4. q.setParameter("i",111);
6. **int** status=q.executeUpdate();
7. System.out.println(status);
8. tx.commit();

### Example of HQL delete query

1. Query query=session.createQuery("delete from Emp where id=100");
2. //specifying class name (Emp) not tablename
3. query.executeUpdate();

### HQL with Aggregate functions

You may call avg(), min(), max() etc. aggregate functions by HQL. Let's see some common examples:

### Example to get total salary of all the employees

1. Query q=session.createQuery("select sum(salary) from Emp");
2. List<Emp> list=q.list();
3. Iterator<Emp> itr=list.iterator();
4. **while**(itr.hasNext()){
5. System.out.println(itr.next());
6. }

### Example to get maximum salary of employee

1. Query q=session.createQuery("select max(salary) from Emp");

### Example to get minimum salary of employee

1. Query q=session.createQuery("select min(salary) from Emp");

### Example to count total number of employee ID

1. Query q=session.createQuery("select count(id) from Emp");

### Example to get average salary of each employees

1. Query q=session.createQuery("select avg(salary) from Emp");

Bottom of Form