1. HIBERNATE – ORM

Any enterprise application performs database operations by storing and retrieving vast amounts of data. Despite all the available technologies for storage management, application developers normally struggle to perform database operations efficiently.

Generally, Java developers use lots of code or proprietary framework to interact with the database. Therefore, it is advisable to use Object Relational Mapping (ORM) to reduct the burden of interacting with the database. ORM forms a bridge between object models (Java program) and relational models (database program) like JDBC.

What is JDBC?

JDBC stands for **Java Database Connectivity**. It 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.	Complex if it is used in large projects.
Good performance with large data.	Large programming overhead.
Very good for small applications.	No encapsulation.
Simple syntax so easy to learn.	Hard to implement MVC concept. Query is DBMS specific.

Why Object Relational Mapping (ORM)?

When we work with an object-oriented system, there is a mismatch between the object model and the relational database table. RDBMSs represent data in a tabular format whereas objectoriented languages, such as Java or C# represent it as an interconnected graph of objects.

Example

One may encounter a few problems when direct interaction takes place between object models and relation database tables. The following example highlights the problems associated with traditional database application structure.

Here we have a Java Class called **Employee** with proper constructors and associated public functions. We have a simple java entity class for an employee having fields (variables) such as id, first_name, last_name, and salary.

```
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 the above object needs to be stored and retrieved using the following RDBMS table. : Following is the Create table statement for Employee class. Its field structure should be same as the given object model (Employee class).

```
create table EMPLOYEE (    id INT
NOT NULL auto_increment,
```

Hibernate

```
   first_name  VARCHAR(20)  default
NULL,      last_name   VARCHAR(20)
default  NULL,     salary       INT
default NULL,
   PRIMARY KEY (id)
);
```

Thee following problems may arise when direction interaction takes place between object models and relation database tables:

- First, what if we need to modify the design of our database after having developed a 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	An 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 an RDBMS represents an association as a foreign key column.
Navigation	The ways you access objects in Java and in 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 the 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 refer 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

2. HIBERNATE – OVERVIEW

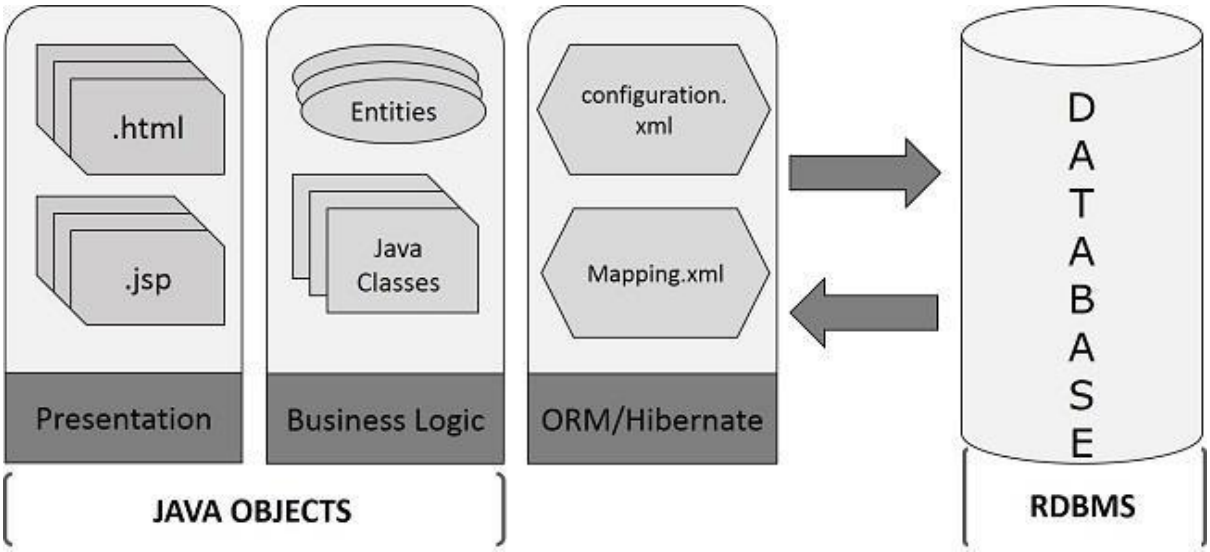
There are several persistent frameworks and ORM options in Java. A persistent framework is an ORM service that stores and retrieves objects into a relational database.

- Enterprise JavaBeans Entity Beans
- Java Data Objects
- Castor
- TopLink
- Spring DAO
- Hibernate, and many more

Hibernate is an **O**bject-**R**elational **M**apping(ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



Hibernate Advantages

Here we have listed down the advantages of using Hibernate:

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code. If there is a change in the database or in any table, then all that you need to change are the XML file properties.
- Provides simple APIs (classes and methods) for storing and retrieving Java objects directly to and from the database.
- Hibernate supports **Inheritance**, **Association relations**, and **Collections**.
- Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Hibernate Supports only unchecked exceptions, so no need to write try, catch, or throws blocks. Generally we have a Hibernate translator which converts Checked exceptions to Unchecked.
- Minimizes database access with smart fetching strategies.
- Hibernate has its own query language. That is **Hiberate Query Language** (HQL) which contains database independent controlers.
- Manipulates Complex associations of objects of your database.
- Hibernate supports caching mechanism: It reduces the number of round trips (transactions) between an application and the database. It increases the application performance.

- Hibernate supports annotations, apart from XML..

Supported Databases

Hibernate supports almost all the major RDBMS database servers. Following is a list of few of the database engines that Hibernate supports:

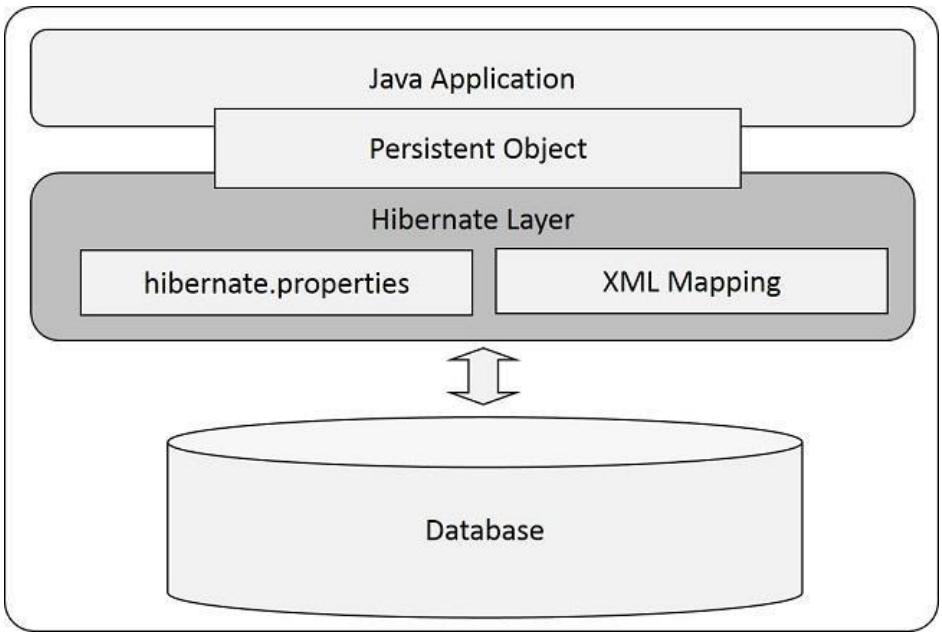
- HSQL Database Engine
- DB2/NT
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

Hibernate supports a variety of other technologies as well including:

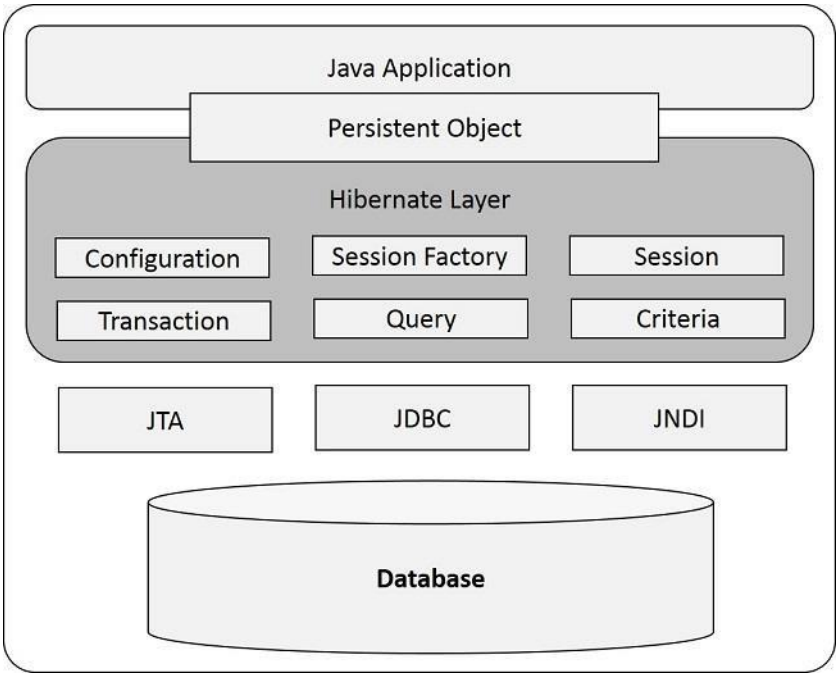
- XDoclet Spring
- J2EE
- Eclipse plug-ins
- Maven

Hibernate has a layered architecture which helps the user to operate without 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 a few important core classes in the Hibernate layer.



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 support Hibernate to be integrated with J2EE application servers.

The following section gives a brief description about the class objects which are involved in the Hibernate Layer of the given architecture diagram. This sectiong gives you a theritical idea of how the hibernate class objects are used to build an application.

Configuration Object

Configuration is a serializable class. It is the first Hibernate object that you need to create in any Hibernate application. It is usually created only once during application initialization. It allows the application to specify properties and mapping documents to be used. The Configuration object provides two keys components:

- **Database Connection:** A database connection is most important for Enterprise and Database applications. It is handled through one or more configuration files supported by Hibernate. Those are **hibernate.properties** file and **hibernate.cfg.xml** file.
- **Mapping Setup:** This component creates the connection between the Java classes and database tables. It creates mapping between each entity java class and each table in the database.

SessionFactory Object

SessionFactory is a Factory Interface used to create **Session** instances. After adding the properties and Mapping files to the Configuration object, it is used to create a SessionFactory object which in turn configures Hibernate (Front-end javaclasses and Back-end tables) for the application. SessionFactory is a thread-safe object and used by all the threads of an application. It is a heavyweight object, usually 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

Session is an Interface that wraps the JDBC connection. That means, it creates a physical connection between the application and 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 Lifecycle of a Session is bounded by the beginning and end of a logical trasaction. It contains three states:

- **transient:** never persistent, currently not associated with any Session.
- **persistent:** currently associated with unique Session.
- **detached:** previously persistent, currently not associated with any Session.

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

Transaction Object

Transaction is an Interface and it 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.

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 is an interface and it is used in **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 is an interface and it is used for retrieving entity data by composing Criterion (Interface) objects. Criterion Objects work like a condition **(WHERE and IF)** in the SQL query, all the criterion objects (conditions) are added to the Criteria Object and that object will be