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 servers to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.

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.



Hibernate Advantages

* Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
* Provides simple APIs for storing and retrieving Java objects directly to and from the database.
* If there is a change in the database or any table, then you need to change the XML file properties only.
* 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.
* Manipulates Complex associations of objects of your database.
* Minimizes database access with smart fetching strategies.
* Provides simple querying of data.
* Hibernate has a layered architecture that 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.



## Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application. It is 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 hibernating**. properties** and **hibernate.cfg.xml**.
* **Class Mapping Setup** − This component creates the connection between the Java classes and database tables.

## SessionFactory Object

The configuration object is used to create a SessionFactory object which in turn 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 a heavyweight object; it is 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

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 do not usually thread-safe and they should be created and destroyed 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 of managing transactions in their 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 execute the query.

Configuration.xml

## Hibernate Properties

**hibernate.dialect**

This property makes Hibernate generate the appropriate SQL for the chosen database.

**hibernate.connection.driver\_class**

The JDBC driver class.

**hibernate.connection.url**

The JDBC URL to the database instance.

**hibernate.connection.username**

The database username.

**hibernate.connection.password**

The database password.

**hibernate.connection.pool\_size**

Limits the number of connections waiting in the Hibernate database connection pool.

**hibernate.connection.autocommit**

Allows auto-commit mode to be used for the JDBC connection

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.

**Transaction beginTransaction()**

Begin a unit of work and return the associated Transaction object.

**Query createQuery(String queryString)**

Create a new instance of Query for the given HQL query string.

**Session get(String entityName, Serializable id)**

Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.

**SessionFactory getSessionFactory()**

Get the session factory that created this session.

**void saveOrUpdate(Object object)**

Either save(Object) or update(Object) the given instance.

**void update(Object object)**

Update the persistent instance with the identifier of the given detached instance.

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Hibernate works best if these classes follow some simple rules, also known as the **Plain Old Java Object** (POJO) programming model.

There are following main rules of persistent classes, however, none of these rules are hard requirements −

* All Java classes that will be persisted need a default constructor.
* All classes should contain an ID to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.
* All attributes that will be persisted should be declared private and have **getXXX** and **setXXX** methods defined in the JavaBean style.
* A central feature of Hibernate, proxies, depends upon the persistent class being either non-final or the implementation of an interface that declares all public methods.
* All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

The POJO name is used to emphasize that a given object is an ordinary Java Object, not a special object, and in particular not an Enterprise JavaBean.

t the mapping elements used in the mapping file −

* The mapping document is an XML document having **<hibernate-mapping>** as the root element, which contains all the **<class>** elements.
* The **<class>** elements are used to define specific mappings from Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database **table** name is specified using the table attribute.
* The **<meta>** element is an optional element and can be used to create the class description.
* The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping type will convert from Java to SQL data type.
* The **<generator>** element within the id element is used to generate the primary key values automatically. The **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence**, or **Hilo** algorithm to create a primary key depending upon the capabilities of the underlying database.
* The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping type will convert from Java to SQL data type.
* Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries, which in turn act on the database.
* Although you can use SQL statements directly with Hibernate using Native SQL, I would recommend using HQL whenever possible to avoid database portability hassles and to take advantage of Hibernate's SQL generation and caching strategies.
* Keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL.