

# Hibernate Framework

## Introduction to Hibernate Framework

**Object Relational Mapping (ORM):** When we work with an object-oriented systems, 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. ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as 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 sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.

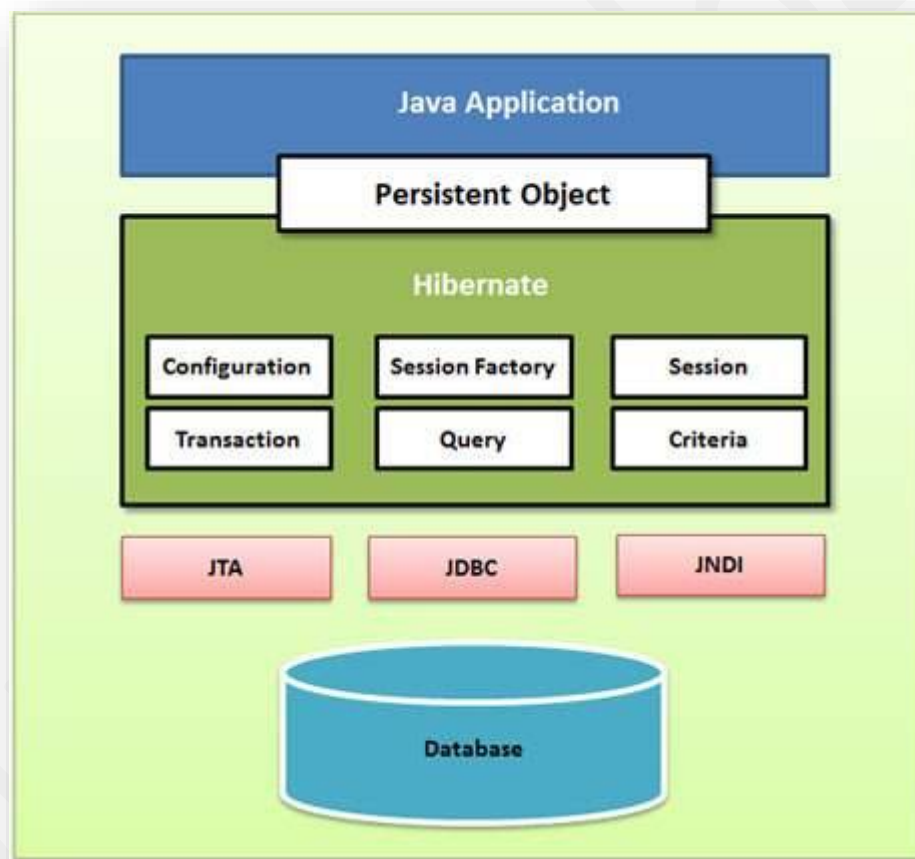


### Hibernate Advantages:

- 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 change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides simple querying of data.

## Hibernate Architecture



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.

## 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 key 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 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 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 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.

## HIBERNATE – PERSISTENT CLASS

The entire concept of Hibernate is to take the values from Java class attributes and persist them to a database table. A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields.

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 in order 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.

# Hibernate Query Language (HQL)

## FROM Clause

```
String hql = "FROM Employee";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## AS Clause

```
String hql = "FROM Employee AS E";  
Query query = session.createQuery(hql);  
List results = query.list();
```

The **AS** keyword is optional and you can also specify the alias directly after the class name, as follows:

```
String hql = "FROM Employee E";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## SELECT Clause

```
String hql = "SELECT E.firstName FROM Employee E";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## WHERE Clause

```
String hql = "FROM Employee E WHERE E.id = 10";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## ORDER BY Clause

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";  
Query query = session.createQuery(hql);  
List results = query.list();
```

If you wanted to sort by more than one property, you would just add the additional properties to the end of the order by clause, separated by commas as follows:

```
String hql = "FROM Employee E WHERE E.id > 10 " +  
            "ORDER BY E.firstName DESC, E.salary DESC ";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## GROUP BY Clause

```
String hql = "SELECT SUM(E.salary), E.firstName FROM Employee E " +  
            "GROUP BY E.firstName";  
Query query = session.createQuery(hql);  
List results = query.list();
```

## Using Named Paramters

```
String hql = "FROM Employee E WHERE E.id = :employee_id";  
Query query = session.createQuery(hql);  
query.setParameter("employee_id", 10);  
List results = query.list();
```

## UPDATE Clause

```
String hql = "UPDATE Employee set salary = :salary " +  
            "WHERE id = :employee_id";  
Query query = session.createQuery(hql);  
query.setParameter("salary", 1000);  
query.setParameter("employee_id", 10);  
int result = query.executeUpdate();  
System.out.println("Rows affected: " + result);
```

## DELETE Clause

```
String hql = "DELETE FROM Employee " +  
            "WHERE id = :employee_id";  
Query query = session.createQuery(hql);  
query.setParameter("employee_id", 10);  
int result = query.executeUpdate();  
System.out.println("Rows affected: " + result);
```

## INSERT Clause

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)" +
            "SELECT firstName, lastName, salary FROM old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

## Aggregate Methods

S.N.	Functions	Description
1	avg(property name)	The average of a property's value
2	count(property name or *)	The number of times a property occurs in the results
3	max(property name)	The maximum value of the property values
4	min(property name)	The minimum value of the property values
5	sum(property name)	The sum total of the property values

The **distinct** keyword only counts the unique values in the row set. The following query will return only unique count:

```
String hql = "SELECT count(distinct E.firstName) FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

## Pagination using Query

S.N.	Method & Description
1	<b>Query <code>setFirstResult(int startPosition)</code></b>  This method takes an integer that represents the first row in your result set, starting with row 0.
2	<b>Query <code>setMaxResults(int maxResult)</code></b>  This method tells Hibernate to retrieve a fixed number <b>maxResults</b> of objects.

Using above two methods together, we can construct a paging component in our web or Swing application. Following is the example which you can extend to fetch 10 rows at a time:

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```