**Hibernate**

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

Hibernate Framework(ORM)

JAVA Object-----------------------------------------------------------------DataBase

[JDBC](https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/)**:** JDBC stands for **Java Database Connectivity**. It is a java application programming interface to provide a connection between the Java programming language and a wide range of databases (i.e), it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

[Hibernate](https://www.geeksforgeeks.org/introduction-to-hibernate-framework/)**:** Hibernate is an open-source, non-invasive, light-weight java ORM(**Object-relational mapping**) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE. It simplifies the interaction of java applications with databases. Hibernate is an implementation of JPA(Java Persistence API).

**JDBC Hibernate**

|  |  |  |
| --- | --- | --- |
| 1. | In JDBC, one needs to write code to map the object model’s data representation to the schema of the relational model. | Hibernate maps the object model’s data to the schema of the database itself with the help of annotations. |
| 2. | JDBC enables developers to create queries and update data to a relational database using the Structured Query Language (SQL). | Hibernate uses HQL (Hibernate Query Language) which is similar to SQL but understands object-oriented concepts like inheritance, association etc. |
| 3. | JDBC code needs to be written in a try catch block as it throws checked exception(SQLexception). | Whereas Hibernate manages the exceptions itself by marking them as unchecked. |
| 4. | JDBC is database dependent i.e. one needs to write different codes for different database. | Whereas Hibernate is database independent and same code can work for many databases with minor changes. |
| 5. | Creating associations between relations is quite hard in JDBC. | Associations like one-to-one, one-to-many, many-to-one, and many-to-many can be acquired easily with the help of annotation |

Dependencies in POM.XML for maven based Hibernate Project

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.1.3.Final</version>

<type>pom</type>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.30</version>

</dependency>

Maven project automatically download all required dependencies for this project

Hibernate project by Simple java application

**You can also use simple java Project to create hibernate project but you need add some jar file for that:**

Search🡪hibernate jars🡪hibernate.org🡪latest stable version download🡪zip archive—extract---lib---required folder—add all jar file

Right click on java project🡪build path🡪configure build path🡪class path🡪add external jars🡪select all jar file-🡪add🡪apply..>apply and close

**Configuring Hibernate:**

**Simply Hibernate is java application establishes communication with your databases to perform different operation on database via java application so for this application need some url,username,password to establish coonection so need to configure it**

**Hibernate.cfg.xml**

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.autocommit"*>false</property>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.cj.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/test </property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>sairam</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"hibernate.hbm2ddl.auto"*>update</property>

<!-- update:only one time table is created if not present otherwise not created if present

<property name=*"hibernate.show\_sql"*>true</property>

<!-- show\_sql will show which SQL query fired by hibernate -->

<property name=*"hibernate.format\_sql"*>true</property>

<!-- format\_sql:will show fired query in well format manner -->

<property name=*"hibernate.current\_session\_context\_class"*>thread</property>

*<!-- 1.A connection pool is used to minimize the number of connections* opened between application and database. Connection pooling means that connections are reused rather than created each time a connection is requested.

2.To facilitate connection reuse, a memory cache of database connections, called a connection pool

3.The Hibernate Connection Pool Size property establishes the number of connections that are permitted between the Model repository and the Model Repository Service database. The default value is 10. -->

<property name=*"hibernate.connection.pool\_size"*>10</property>

</session-factory>

</hibernate-configuration>

**Hibernate Architecture :**

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

* Java application layer---java application
* Hibernate framework layer---session factory,transaction factory
* Backhand api layer---jdbc
* Database layer---mysql

Java Class-----Persistent Object---------------------------- ------------

Hibernate Database

**Configuration:**

* Configuration is a class which is present in org.hibernate.cfg package. It activates Hibernate framework. It reads both configuration file and mapping files.(hibernate.cfg.xml) which contains all databae related configuration or validation

It activate Hibernate Framework

**Configuration cfg=new Configuration();**

It read both cfg file and mapping files

**cfg.configure(“hibernate.cfg.xml”);**

**SessionFactory:**

* SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
* It is immutable and thread-safe in nature, It is built only once at the start of an application. Once we have a session, we can do all the thread operations – create, delete, modify, read, update and other.

buildSessionFactory() method gathers the meta-data which is in the cfg Object. From cfg object it takes the JDBC information and create a JDBC Connection.

**SessionFactory factory=cfg.buildSessionFactory();**

**Session:**

* Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.
* It opens the Connection/Session with Database software through Hibernate Framework.
* It is a light-weight object and it is not thread-safe.
* Session object is used to perform CRUD operations.

**Session session=factory.buildSession();**

**Transaction:**

* Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
* Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.

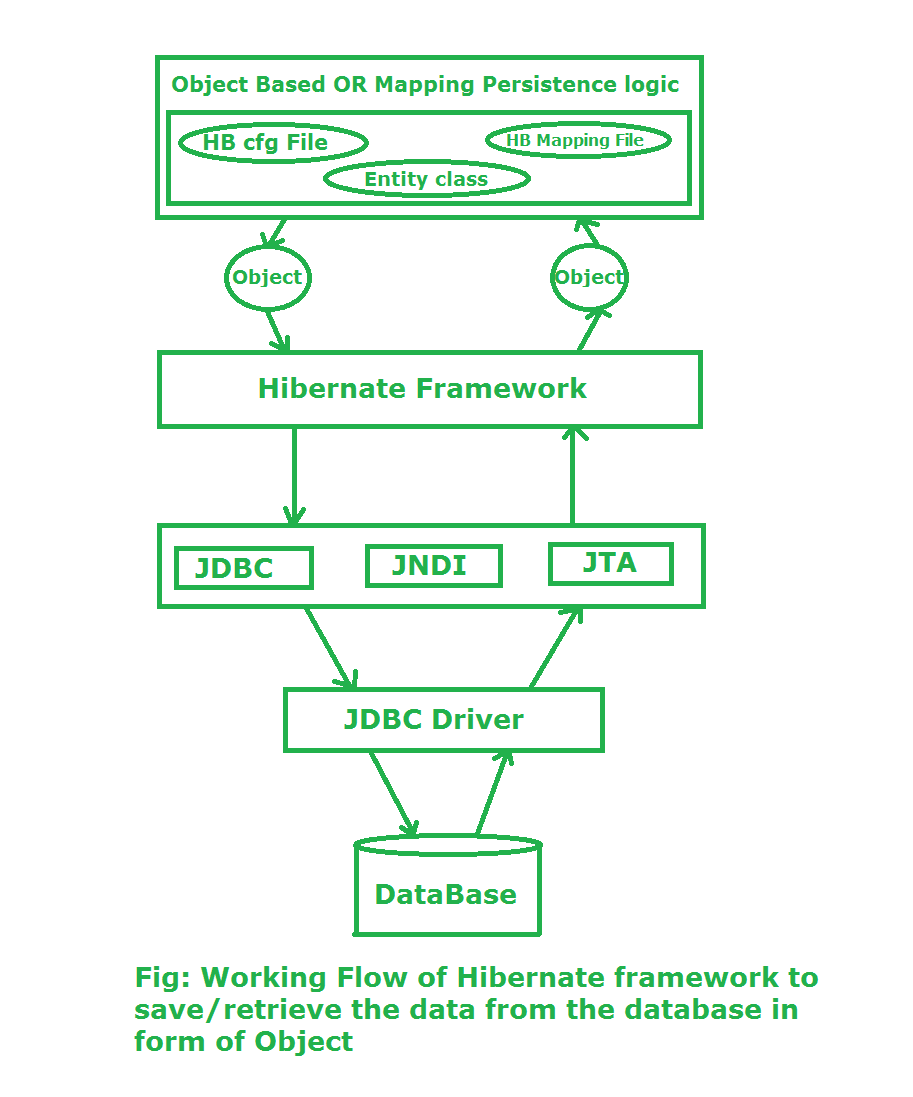
**Transaction tx=session.beginTransaction();**

**tx.commit();**

**Query:**

* Query is an interface that present inside org.hibernate package.
* A Query instance is obtained by calling Session.createQuery().
* This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():
  1. A particular page of the result set may be selected by calling setMaxResults(), setFirstResult().
  2. Named query parameters may be used.

**Query query=session.createQuery();**



**Stage I:** In first stage, we will write the persistence logic to perform some specific operations to the database with the help of Hibernate Configuration file

**Stage II:**In second stage, our class which contains the persistence logic will interact with the hibernate framework where hibernate framework gives some abstraction do perform some task.

**Stage III:**In third stage, our hibernate framework interact which JDBC, JNDI, JTA etc to go to the database to perform that persistence logic.on database

**Stage IV & V:**In fourth & fifth stage, hibernate is interact with Database with the help of JDBC driver. Now here hibernate perform that persistence logic which is nothing but **CRUD** operation. If our persistence logic is to retrieve an record then in the reverse order

We can create Hibernate Application using two approaches

1-XML based approach

2-Annotation based approach

Annotations in Hibernate :

**@Entity:**This annotation marks this class as an entity and this class mapped in database

**@Table:**This annotation specifies the table name where data of this entity is to be persisted. If you don't use @Table annotation, hibernate will use the class name as the table name by default.

**@Id** :This annotation marks the Primary key for this entity which mapped in database

**@GeneratedValue**: This annotation will generate value for column automatically to need to set value manually

**@Column:**If you want to change column name associated with Entity which mapped in database then this annotation used

**@Lob:**This annotation tells the object is large object not simple object

***Other-@OneToOne,@OneToMany,@ManyToOne and @JoinColumn***

**@Embeddable**:when we want to embed one class to other class without creation of Entity table then we use @Embeddable annotation over other class

**Hibernate LifeCycle :-**

Transient state

* The transient state is the initial state of an object.
* Once we create an instance of POJO class, then the object entered in the transient state.
* Here, an object is not associated with the Session. So, the transient state is not related to any database.
* Hence, modifications in the data don't affect any changes in the database.
* The transient objects exist in the heap memory. They are independent of Hibernate.

Persistent state

* As soon as the object associated with the Session, it entered in the persistent state.
* Hence, we can say that an object is in the persistence state when we save or persist it.
* Here, each object represents the row of the database table.
* So, modifications in the data make changes in the database.

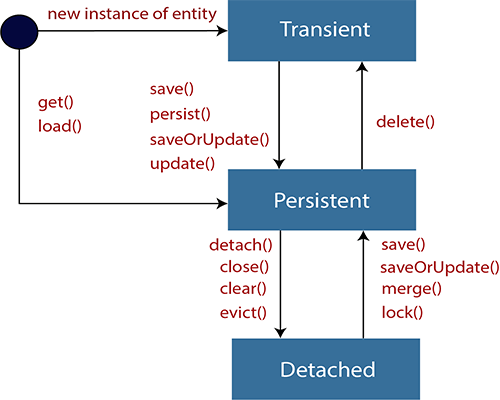
Detached State

* Once we either close the session or clear its cache, then the object entered into the detached state.
* As an object is no more associated with the Session, modifications in the data don't affect any changes in the database.
* However, the detached object still has a representation in the database.

Transient------------------🡪Persistence-----------------------🡪Detached

S &D not asso S&D asso S is not ass but asso with D

S=session and D=Database



**Caching in Hibernate:**mechanism used to enhance performance of java app.

Used to reduce interaction of java app with your Database

**Without Caching :**

Java Regular Database

App framework

Problem without caching is when multiple times we try to execute same query with java app on database each time it interact with database for same query so it reduce performance of java app

**With Caching :**

Java Hibernate Database

App framework (reduce interaction of java app with DB)

Query result store in cache

Cache

Problem without caching solved by with cache because whenever we try toexecute same query multiple time on DB only first time it interact with DB and save the result in cache after that for same query it check wheather result present in cache if yes then it retrieve without interaction

**Two Types of Caching**

**1.First level caching**

**2.Second level caching**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Basic | First level cache is a session level cache and it is always associated with session level object | Second level cache is session factory level cache and it is available across all sessions |
| 2 | Enabled | It is enabled by default. | It is not enabled by default. |
| 3 | Availability | It is available for a session | It is available across all session. |
| 4 | Configuration | No Extra configuration required | We have to decide  which concurrency strategy to use and also need to configure cache expiration and physical cache attributes. |

**How To Enable Second Level Cache :**

Need to download some dependencies---

**Step 1)**search ehcache maven ----open depend from official site copy and paste in pom.xml

**<!-- https://mvnrepository.com/artifact/net.sf.ehcache/ehcache -->**

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache</artifactId>

<version>2.10.9.2</version>

</dependency>

**Step2)** search hibernate cache maven –open official site

**<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>5.6.10.Final</version>

<!-- hibernate ecache version should match with hibernate core version -->

</dependency>

**Step 3)**Enable second level cache in hibernate.cfg.xml file need some property

<property name=”cache.use\_second\_level\_cache”>true</property>

<property name=”cache.region.factory\_class”>net.sf.ehcache.hibernate.EhCacheRegionFactory

</property>

**Some Annotations:**

**@Transient :**This annotation indicated field not to be persisted or ingnored to save in database

**@Enumerated** :maps given number with enum

**☺ Hibernate Query Language (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.

With the help of HQL you can execute some complex query on database based on some condition and performed CRUD operation by some criteria .

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

Hibernate supports HQL and SQL you can use either but better to use HQL it is similar to SQL having small difference.

HQL is database independent whereas SQL is Database independent

**Query query=session.createQuery(“query wrote here…”);**

**query.executeUpdate();**

1. **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
2. **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

**With this we can perform operation on database by some criteria.**

**Example :**

Query q=session.createQuery("update User set name=:n where id=:i");

// update User set name=’Akshay’ where id=101 //set var value inside query

// update User u set u.name=:n where u.id=:i // ---alias

q.setParameter("n","Udit Kumar");  //set var value outside query

q.setParameter("i",111);

**Jpql and HQL are object model focused query language similar in nature of SQL query, JPQL is subset of HQL, so JPQL query is valid in HQL**

**Mapping in Hibernate :**

**1.OneToOne Mapping**

OneToOne mapping two Entity linked together either unidirectionally or bidirectionally

Ex :One Question having only one Answer

@OneToOne

Entity1 --------------------------------------🡪Entity2

**Unidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@OneToOne**

**@JoinColumn(name=”col\_name”)**

**Entity2 entity2**

**} }**

**Bidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@OneToOne @OneToOne(mappedBy=)**

**@JoinColumn(name=”col\_name”) Entity1 entity1;**

**Entity2 entity2**

**}**

**mappedBy=”entity2” :if you do not use this attribute then foreign key column created in each table which refer irrespective to avoid that we use mappedBy i.e only one foreign key created Entity1 which refer Entity2**

**2.OneToMany mapping**

One Entity value can refer to multiple other Entity value either unidirectionally and bidirectinally

Ex-One Question having multiple values

Entity1--------------------------------------------------------🡪Entity2

**Unidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@OneToMany**

**@JoinColumn(name=”col\_name”)**

**List<Entity2> entity2**//many values stored

**} }**

**Bidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@OneToMany @ManyToOne(mappedBy=)**

**@JoinColumn(name=”col\_name”) List<Entity1> entity1;**

**List<Entity2> entity2**

**} }**

**2.ManyToMany mapping**

Many Entity value can refer to multiple other Entity value either unidirectionally and bidirectinally

Ex-One Question having multiple values

Entity1 Entity2

Ex-Multiple project assigned to multiple Employee

**Unidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@ManyToMany**

**@JoinColumn(name=”col\_name”)**

**List<Entity2> entity2**//many values stored

**} }**

**Bidirectinal**

**@Entity @Entity**

**Class Entity1{ class Entity2{**

**@Id @Id**

**Int q\_id; int a\_id;**

**String que; String ans;**

**@ManyToMany @ManyToMany(mappedBy=)**

**@JoinColumn(name=”col\_name”) List<Entity1> entity1;**

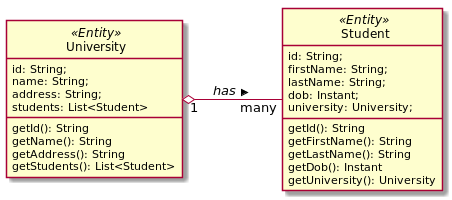
**List<Entity2> entity2**

**} }**

Fetch Type :

Sometimes you have two entities and there's a relationship between them. For example, you might have an entity called University and another entity called Student and a University might have many Students:

The University entity might have some basic properties such as id, name, address, etc. as well as a collection property called students that returns the list of students for a given university:

[](https://i.stack.imgur.com/N1PL3.png)

Now when you load a University from the database, JPA loads its id, name, and address fields for you. But you have two options for how students should be loaded:

1. To load it together with the rest of the fields (i.e. eagerly), or
2. To load it on-demand (i.e. lazily) when you call the university's getStudents() method.

When a university has many students it is not efficient to load all of its students together with it, especially when they are not needed and in suchlike cases you can declare that you want students to be loaded when they are actually needed. This is called lazy loading.

**Here's an example, where students is explicitly marked to be loaded eagerly:**

@Entity

public class University {

@Id

private String id;

private String name;

private String address;

@OneToMany(fetch = FetchType.EAGER)//load all student eagerly

private List<Student> students;

}

**And here's an example where students is explicitly marked to be loaded lazily:**

@Entity

public class University {

@Id

private String id;

private String name;

private String address;

@OneToMany(fetch = FetchType.LAZY)//load student as per demand

private List<Student> students;

}

| **Sr. No.** | **Key** | **Lazy** | **Eager** |
| --- | --- | --- | --- |
| 1 | Fetching strategy | In Lazy loading, associated data loads only when we explicitly call getter or size method. | In Eager loading, data loading happens at the time of their parent is fetched |
| 2 | Default Strategy in ORM Layers | ManyToMany and OneToMany associations used lazy loading strategy by default. | ManyToOne and OneToOne associations used lazy loading strategy by default. |
| 3 | Loading Configuration | It can be enabled by using the annotation parameter :  fetch = FetchType.LAZY | It can be enabled by using the annotation parameter :  fetch = FetchType.EAGER |
| 4 | Performance | Initial load time much smaller than Eager loading | Loading too much unnecessary data might impact performance |

**Cascading :**

Cascading is a feature in Hibernate, which is **used to manage the state of the mapped entity whenever the state of its relationship owner (superclass) affected**. When the relationship owner (superclass) is saved/ deleted, then the mapped entity associated with it should also be saved/ deleted automatically.

**Spring Framework with Hibernate integration(using maven):**

Advantages of ORM framework with spring :

1.Less coding.

2.Easy to test

3.No need to handle exception explicitly.

4.Integrated transaction management

**Why ?**

**1.No need to define hibernate.cfg.xml file separately.**

**2.We provide all configuration in spring-config.xml file**

**3.Hibernate Steps :**

Configuration cfg=**new** Configuration();

cfg.configure("hibernate.cfg.xml");

SessionFactory factory=cfg.buildSessionFactory();

Session session=factory.openSession();

Transaction tx=session.beginTransaction();

We no need to follow these steps when spring framework integrate with Hibernate because spring having predefine HibernateTemplate class we simply use its reference.

**Steps to create spring app with hibernate or Spring ORM application**

**Step 1)**New🡪project🡪maven project🡪simple project🡪(groupId,artifactId,name,jar file)--🡪finish

**Step2)**open pom.xml🡪add spring core,spring context,mysql-connector and very important spring-orm dependencies due to this we able to use HibernateTemplate class reference to perform operation on Database

*Spring ORM release==Spring context release*

**Also add hibernate-core and spring orm dependencies in pom.xml**

<!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>5.3.14</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.14.Final</version>

</dependency>

**Step3)** update project

**Step4)**create spring-config.xml file in src/main/resources for adding spring and hibernate configuration

<bean

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*

name=*"ds"*>

<property name=*"driverClassName"*

value=*"com.mysql.cj.jdbc.Driver"*></property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/springjdbc"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"sairam"*></propert>

</bean>

<bean

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*

name=*"localfactory"*>

<property name=*"dataSource"* ref=*"ds"*></property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQL57Dialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

</props>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>com.app.pojo.Student</value>

</list>

</property>

</bean>

<bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*

name=*"hibernateTemp"*>

<property name=*"sessionFactory"*>

<ref bean=*"localfactory"*></ref>

</property>

</bean>

<bean class=*"com.app.Dao.DaoClass"* name=*"daoRef"*>

<property name=*"hibernateTemp"*>

<ref bean=*"hibernateTemp"*></ref>

</property>

</bean>