Hibernate

What is Hibernate?

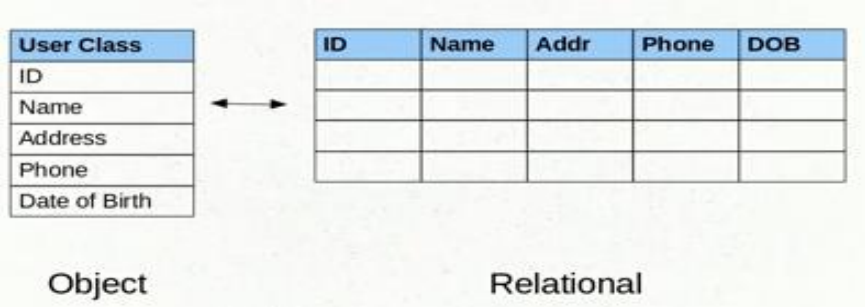
Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate and retrieve data from the database.

Hibernate is a framework that can be used for object-relational mapping intended for Java programming language. More specifically, it is an ORM (object-relational mapping) library that can be used to map object-relational model in to conventional relational model.

In simple terms, it creates a mapping between Java classes and tables in relational databases, also between Java to SQL data types. Hibernate can also be used for data querying and retrieving by generating SQL calls. Therefore, the programmer is relieved from the manual handling of result sets and converting objects.

The ORM tool internally uses the JDBC API to interact with the database. Hibernate provide the abstraction layer over the JDBC.

ORM is the concept according to that database tables are mapped with the classes in the object oriented program.



- Object's Property **ID** set to **ID** column
- Object's Property **Name** set to **Name** Column
- Object's Property **Address** set to **Address** column
- Object's Property **Phone** set to **Phone** Column
- Object's Property **Date of Birth** set to **DOB** column

Why, ORM Tools like Hibernate are used?

1. Hibernate mainly solves the object-relational impedance mismatch problems that arise when a relational database is connected by an application written in an object-oriented programming language style. Object-relational impedance mismatches arise due to data type differences, manipulative differences, transactional differences, and structural and integrity differences. It automatically maps the domain object in the relational database.
2. Hibernates code is database independent because you do not need to change the HQL queries when you change databases like MySQL, Oracle, etc. Hence, it is easy to migrate to a new database.
3. Hibernate minimizes code changes when we add a new column to a database table.

What does hibernate?

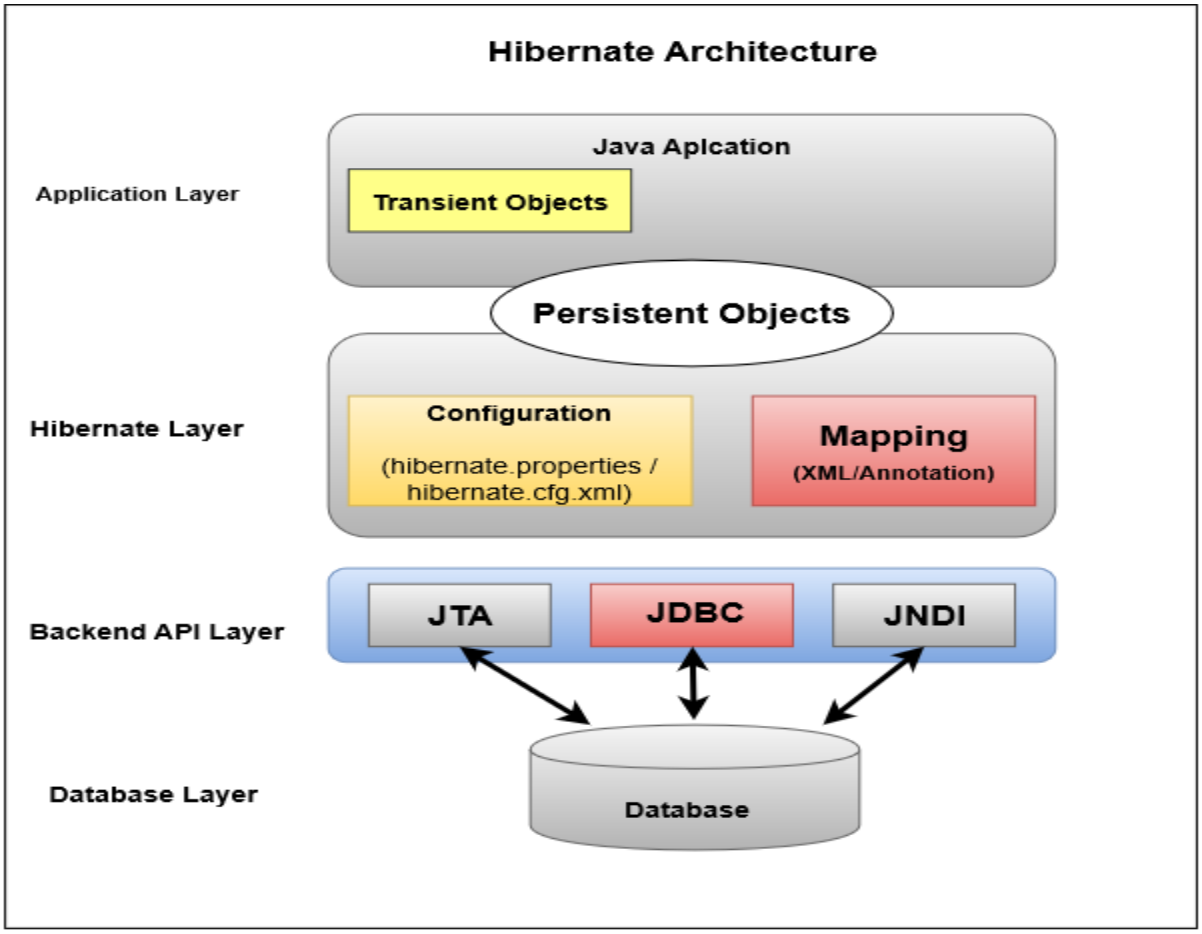
- Map Java class to database tables & vice versa
- Data query and retrieval facility
- Generate the SQL query based on the underline DB. And attempts to relieve the developer from manual result set handling and object conversion.
- Make application portable to all relational DB.
- Enhance performance by providing the different levels of cache (First, Second and Query level).

What is the difference between JPA and Hibernate?

JPA is a framework for managing relational data in Java applications, while Hibernate is a specific implementation of JPA (so ideally, JPA and Hibernate cannot be directly compared). In other words, Hibernate is one of the most popular frameworks that implements JPA. Hibernate implements JPA through Hibernate Annotation and EntityManager libraries that are implemented on top of Hibernate Core libraries. Both EntityManager and Annotations follow the lifecycle of Hibernate. The newest JPA version (JPA 2.0) is fully supported by Hibernate 3.5. JPA has the benefit of having an interface that is standardized, so the developer community will be more familiar with it than Hibernate. On the other hand, native Hibernate APIs can be considered more powerful because its features are a superset of that of JPA.

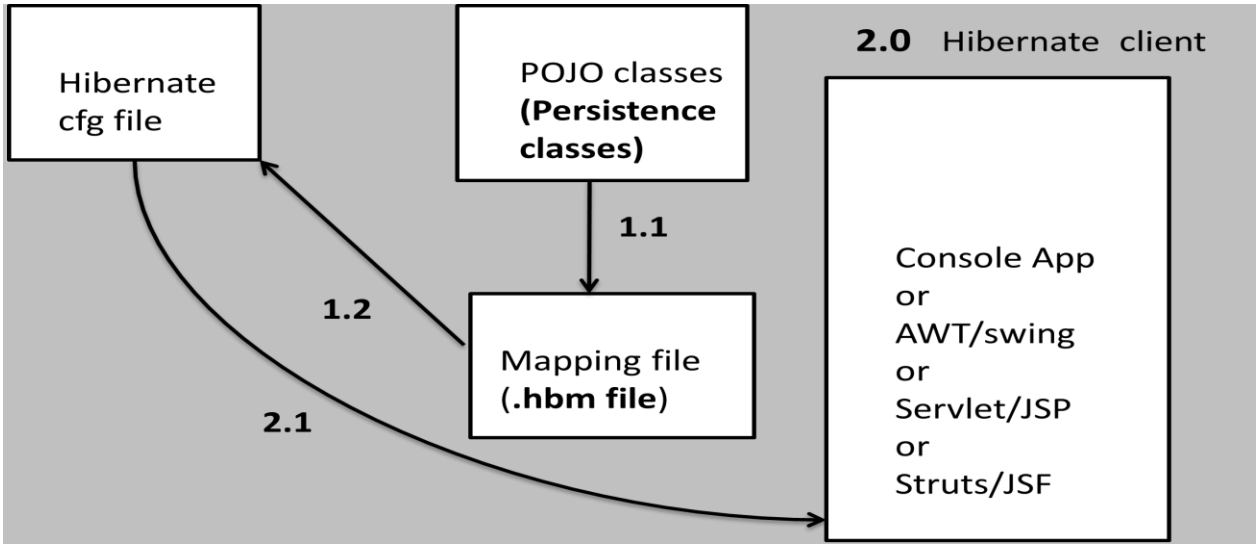
Hibernate Architecture:

In Hibernate, data will be transferred from Java Application to the Database in four Layers.



1. **Java Application Layer:** In Hibernate, all the data will be transferred from the Java Application to Database using the Hibernate Persistence object.
2. **Hibernate Layer:** Hibernate contains Mapping properties and Configuration Properties.
3. **Internal backend API Layer:** Hibernate internally uses existing Java API like JDBC API, JTA (Java Transaction API), JNDI (Java Naming and Directory Interface).
    - JDBC provides operations on relational Databases (Abstraction), Hibernate supports any type of database and JDBC Driver.
    - If needed to create any Connection pool Object at that time JNDI is used.
    - Whenever the data needs to transfer from Java Application to Database then JTA is used. Hibernate is integrated with J2EE application servers.
4. **Database Layer**

Components of the hibernate application



1.1: - Information about the persistence class written in the mapping file.

1.2: -Name of the mapping file written into the configuration file.

2.0: -Client application started.

2.1: -Configuration retrieved in the client application to perform the database operation based on the POJO classes.

1. **Hibernate POJO classes:**
   JavaBeans: JavaBeans are reusable software components for Java that can be manipulated visually in a builder tool. Practically, they are classes written in the Java programming language conforming to a particular convention. A JavaBean is a Java Object that is serializable, has a nullary constructor, and allows access to properties using getter and setter methods.
   POJO Class in Java: POJO stands for Plain Old Java Object. A POJO class is just a normal Java class, which don't extend and not implementing to the technologies specific class and API and that java class is known as the POJO classes.

1. All properties must public setter and getter methods
2. All instance variables should be private
3. Should not Extend prespecified classes.
   i. public class Foo extends HttpServlet {...}
4. Should not Implement prespecified interfaces.
   i. public class Bar implements EntityBean {...}
5. Should not contain prespecified annotations.
   i. @javax.persistence.Entity public class Baz { …}
6. It may not have no argument constructor

The Hibernate POJO classes created by the application developer these classes are used to represent the database table in the java application.

2. **Hibernate Persistent Classes:** 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.
    1- 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.
    2- All attributes that will be persisted should be declared private and have getXXX() and setXXX() methods defined in the JavaBean style.
    3- 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.
    4- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.
    5- 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.

Example:

```
public class Employee {

	private Long id;
	private String name;
	private String contact;
	private String address;
	private Double salary;

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getContact() {
		return contact;
	}
	public void setContact(String contact) {
```

```
		this.contact = contact;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public Double getSalary() {
		return salary;
	}
	public void setSalary(Double salary) {
		this.salary = salary;
	}

}
```

Client application of the hibernate maintain the hibernate persistence class object (POJO class Objects) in three states-

A. Transient state:

- Whenever an object of a POJO class is created for the first time using the new () operator, then it will be in the **Transient state.**
- When the object is in a Transient state **it doesn't represent any row of the database.**
- If we modify the data of a POJO class object, when it is in transient state then **it doesn't effect on the database table.**

B. Persistence state:

- The state of an object is being associated with hibernate session is called as Persistent state. The Persistent object represents the database entity and it having a unique identifier value, which represents the database table.
- The values associated with the persistent object are sync with the database. It means, if we change the values in persistent state objects, the changes will automatically get effected in the database, no need to execute insert/update operations.

C. Detached state:

- Once we close the Hibernate Session, the persistent instance will become a detached instance.
- A detached instance is an object that has been persistent, but its Session has been closed. The reference to the object is still valid, of course, and the detached instance might even be modified in this state. A detached instance can be reattached to a new Session at a later point in time, making it (and all the modifications) persistent again. This feature enables a programming model for long running units of work that require user think-time. We call them application transactions, i.e., a unit of work from the point of view of the user.

3. **The Hibernate Mapping File:** The mapping between Java POJO class and the database tables is provided using either XML or annotation.

   XML Mapping:

   - These are the XML base file that contains a mapping between the POJO classes (hibernate persistence classes) and the database table.
   - Mapping files also contains the relationship between the database tables in the form of relationship between the java classes.
   - There is no default name for the Hibernate mapping file. Conventionally the name of mapping file should be same as the name of persistence class.
   - This file guides the Hibernate towards generating the internal JDBC code. In one application we can have more than one Hibernate mapping files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

        <class name="com.xmlconfig.firstapp.model.Employee"
                table="Employee">
                <id name="id" column="id">
                <generator class="increment"/>
                </id>
                <property name="name" column="name" />
                <property name="contact" column="contact" />
                <property name="address" column="address" />
                <property name="salary" column="salary" />
        </class>
</hibernate-mapping>
```

Hibernate identity generator: -

In the Hibernate there is the number of algorithms to generate the identity value of the record. In that case no need to specify the identity value during the insertion of the record.

```
<hibernate-mapping>
        <class name="Employee" table="employee">
                <id name="id" column="id">
                        <generator class="assigned" />
                </id>
                <property name="name" column="name" />
                <property name="address" column="address" />
        </class>
</hibernate-mapping>
```

The **<generator>** sub element of **<id>** used to generate the unique identifier for the objects of persistent class. There are many generator classes defined in the Hibernate Framework.

List of generators:

- Hibernate using different primary key generator algorithms, for each algorithm internally a class is created by hibernate for its implementation.
- Hibernate provided different primary key generator classes and all these classes are implemented from **hibernate.id.IdentifierGeneratar** Interface.

The following are the list of main generators we are using in the hibernate framework

assigned:

- This is the default generator class used by the hibernate, if we do not specify <generator > element under id element then hibernate by default assumes it as "assigned"
- If generator class is assigned, then the programmer is responsible for assigning the primary key value to object which is going to save into the database

increment:

- First select the max id if there, if no 1 as max
- For each record it increments by 1 ((maximum value +1))
- This algorithm generates the id value of type long, short, or int.
- This algorithm works with all the databases

identity:

- First select the max id if there, if no 1 as max
- For each record it increments by 1 (maximum value +1)
- This algorithm generates the id value of type long, short, or int.
- This algorithm is **not suitable for oracle** it can work for MYSQL, DB2, etc.

sequence:

- This generator class is database dependent it means, we cannot use this generator class for the entire database, we should know whether the database supports sequence or not before we are working with it.
- While inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record.
- If programmer has created a sequence in the database then that sequence name should be passed as the generator.

```
<id name="id" column="id">
        <generator>
                <param name="sequence">my_sequence</param>
        </generator >
</id>
```

- This algorithm generates the id value of type long, short, or int.
- This algorithm is **not suitable for MySQL** it can work for Oracle, DB2, PostgreSQL, etc.

4. **Hibernate Configuration File**: When the Hibernate client gets started first of all there is the requirement to achieve the Hibernate environment by loading the Hibernate configuration file.

   org.hibernate.cfg.Configuration:-An instance of this class is used to load the Hibernate Configuration file. The configuration object is used to create the session factory object and then session factory is used to create the session object.

   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.** By default, it is placed under src/main/resource folder hibernate.cfg.xml file contains database related configurations and session related configurations.
   - **Class Mapping Setup:** This component creates the connection between the Java classes and database tables.

```
<? xml version='1.0' encoding='utf-8'?>
<! DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <! -- Database Properties connection start -->
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost:3306/mydb</property>
      <property name="connection.username">root</property>
      <property name="connection.password">root</property>
         <! -- Database Properties connection start -->

      <! – Related to Hibernate Properties Start-->
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="show_sql">true/false</property>
<     <property name="sql_format">true/false</property>
<     <property name="use_sql_comment">true/false</property>
      <property name="hbm2ddl.auto">validate/create/update/create-drop</property>
         <! – Related to Hibernate Properties end-->

<! -- Mapping file name(s)-->
      <mapping resource="Employee.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

Configuration file contains 3 types of information.

1- **Connection Properties:** We specify JDBC class driver name (for MySQL, Oracle etc.), connection URL (which also specifies host name, port number, and database name), user name and password to connect to the database.

2- **Hibernate Properties:**

- **Hibernate Dialects:**

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

For connecting any hibernate application with the database, you must specify the SQL dialects. There are many Dialects classes defined for RDBMS in the **org.hibernate.dialect** package. They are as follows:

Oracle (any version)	`org.hibernate.dialect.OracleDialect`
Oracle9i	`org.hibernate.dialect.Oracle9iDialect`
Oracle10g	`org.hibernate.dialect.Oracle10gDialect`
MySQL	`org.hibernate.dialect.MySQLDialect`

- **show_sql :** If the value is true, We can see generated SQL statements in console.

```
<property name="show_sql">true</property>
```

- **format_sql**: If the value is true, we can see generated SQL statements in a readable format.

```
<property name="format_sql">true</property>
```

- **use_sql_comments:** If the value is true, we can see comments in generated SQL statements.

```
<property name="use_sql_comment">true</property>
```

- **hibernate.hbm2ddl.auto in Hibernate:** hibernate.hbm2ddl.auto is automatically validates and exports DDL to schema when the sessionFactory is created.

  By default, it is not doing any creation or modification automatically on db. If user sets below values then it is doing DDL schema changes automatically.

```
<property name="hbm2ddl.auto"> validate / create / update / create-drop </property>
```

  - **create** - If the value is CREATE then hibernate first drops the existing table, then creates a new table and then executes operations on newly created table. The only problem with the value **"create"** is, we lose existing table data.

- **update**- If the value is update then, Hibernate checks for the table and columns. If table doesn't exist then it creates a new table and if a column doesn't exist it creates new column for it. But in the case of value **"update"** hibernate doesn't drop any existing table, here we don't lose existing table data.
- **validate**- validate existing schema: If the value is validated then hibernate only validates whether the table and columns are existing or not. If the table doesn't exist then hibernate throws an exception.
- **create-drop-** If the value is create-drop then, hibernate first checks for a table and do the necessary operations and finally drops the table after all the operations are completed. The value **"create-drop"** is given for unit testing the hibernate code.

3- **Mapping file name(s):** mapping tags are used to specify the mapping file where we define the mapping between java class and database table.

```
<mapping resource="Employee.hbm.xml"/>
```

5. **Hibernate Client application:**

While developing the persistence logic in the hibernate client application we need to create two important object-

- SeseionFactory object
- Session object.

SessionFactory: - It will be created based on the entry of hibernate configuration file by using the **org.hibernate.cfg.Configuration** class object.

Point about SessionFactory -

- This object represents a JDBC connection pool for the one database.
- The SessionFactory objects also represent to the cache for the retrieved record from the database.
- SessionFactory object is the object of java class that implements SessionFactory interface.
- This object is capable of creating one or more hibernate session Object.

Example: Hibernate 5

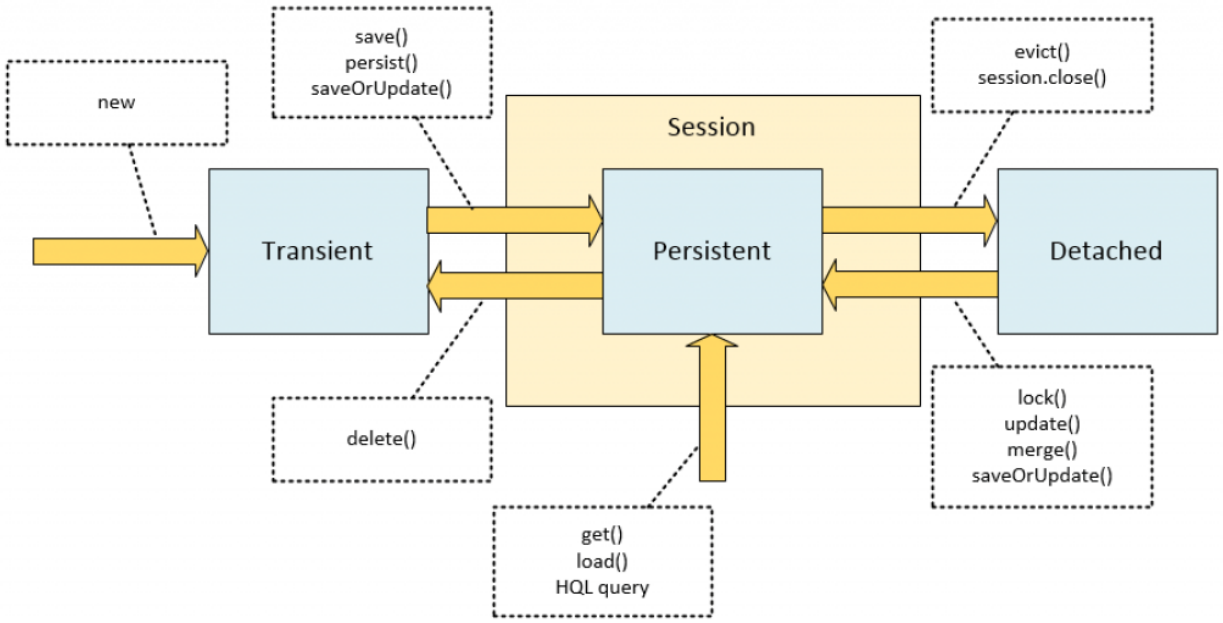
```
public static SessionFactory getSessionFactory() {
		if (sessionFactory == null) {
			try {

	//configure the registry from a resource lookup for a cfg.xml config file
				 //Create registry
registry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
				// Create MetadataSources
				MetadataSources sources = new MetadataSources(registry);
				// Create Metadata
				Metadata metadata = sources.getMetadataBuilder().build();
				// Create SessionFactory
			sessionFactory = metadata.getSessionFactoryBuilder().build();
			} catch (Exception e) {
				e.printStackTrace();
				if (registry != null) {
					StandardServiceRegistryBuilder.destroy(registry);
				}
			}
		}
		return sessionFactory;
	}
```

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, retrieved and delete through a Session object.

Instances may exist in one of three states:

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



public interface Session extends Serializable: the main runtime interface between a Java application and Hibernate. This is the central API class abstracting the notion of a persistence service. The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes.

List Hibernate Session interface methods:

1. **Transaction beginTransaction():** Begin a unit of work and return the associated Transaction object.

   A Transaction is associated with Hibernate Session and instantiated by calling the **session.beginTransaction()**

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

   The methods of Transaction interface are as follows:

   - **void begin():** starts a new transaction**.**
   - **void commit() :** Flush the associated Session and end the unit of work.
   - **void rollback():** forces this transaction to rollback. (In case of any issues, call rollback() error on the transaction object.)

Insert the Record

2. **Serializable save(Object object):** Make a transient object to persistent state. This method stores the given object in the database. Before storing, it checks for generated identifier declaration and process it first, then it will store into DB.
3. **void persist(Object object):** Make a transient object to persistent state. This operation cascades to associated instances if the association is mapped with cascade="persist".

   Difference between save() and persist():-

S. No	save()	persist()
1	**session.save(e)** First generate the identity value for the object then insert the record and also returns the generated identity value.	**session.persist(e)** method also generated identity value for the object but never returns the identity value.
2	**save()** is only supported by Hibernate	**persist()** is supported by both JPA and Hibernate
3	**save()** method returns an identifier so that an insert query is executed immediately to get the identifier, no matter if it are inside or outside of a transaction boundaries. The save() method is not good in a long-running conversation with an extended Session context.	**persist()** also guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. This is useful in long-running conversations with an extended Session/persistence context.

Updating the record:-

4. **public Object merge (Object object) throws HibernateException:** It merge the detached object data into persistent data.
5. **public void update(Object object):**
6. **public void saveOrUpdate(Object object) throws HibernateException:**
   - If the record is not present in the database, it will call save() method and inserts the record in the database.
   - If the record is present in the database, it will call update() method and updates the record in the database.

Difference between merge () and update ():-

S.No	merge ()	update ()
1	It will update the record if exist otherwise insert the new record.	It update the record if exist otherwise throws the HibernateException.
2	**merge()** returns the persistent state object.	**update()** method update the record but does not return the persistence state object.

Difference between merge() and saveOrUpdate() method:-

Both method perform the updating or insertion of the record merge () method returns the persistence state object and saveOrUpdate() method does not return.

Delete the record:-

public void delete(Object object): session.delete() method is used to delete the record. Remove a persistent instance from the data store. This operation cascades to associated instances if the association is mapped with **cascade="delete".**

Selecting the record from the database table:

Hibernate Session provides two methods to access the object (session.get() & session.load()).

7. **Object load(Class class, Serializable id):**

   The load()method of the Session class use to select the record from the database table.

   There are two overloaded load() method :-

   public Object load(class name of POJO class, Serializable id)

   public void load(object Pojo object, serializable id).

S.No	load(class)-lazy loading	load(object)-eager loading
1	It selects the record from the table and creates the object of the specified POJO class and returns the reference id of the object.	In this load() method we pass the object and this method will select the record from the table and populate the data into specified object.
2	It performs the lazy loading.	It performs the eager loading
3	In case of lazy loading the record will be retrieved when any getter method is invoked or need to use object.	In case of eager loading load method will immediately execute the select query as the load method invoked.

8. **public Object get(class pojo class, Serializable id):** This method is also used to get the record from the database. It also used to eager loading.

The difference between load(object ob) and get(class, id)

S.No	load()	get()
1	Use load() method when you are sure that the object exists in database.	Use get() method if you are not sure that the object exists in database.
2	load() method throws hibernate.ObjectNotFoundException (which is Un-checked exception) if object not found in cache as well as in database.	get() method returns NULL , if object is not found in cache as well as in database

9. **public void flush():**Force this session to flush. Must be called at the end of a unit of work, before committing the transaction and closing the session. Flushing is the process of synchronizing the underlying persistent store with persistable state held in memory.
10. **public void refresh(object POJO Object):** This method performs the work opposite to flush method. The refresh method instruct to the hibernate software to reload the record into persistence state object. This is nothing but perform the synchronization of POJO object with the table record.
11. **Connection close ():** End the session by releasing the JDBC connection and cleaning up.