1.Hibernate CRUD operations.

Employee pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.data</groupId>
<artifactId>Emp_DB_Maven_prj</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>

        <!-- Hibernate 4.3.6 Final -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>4.1.6.Final</version>
</dependency>
<!-- Mysql Connector -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.28</version>
</dependency>
</dependencies>
```

Employee hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name = "hibernate.dialect">
org.hibernate.dialect.MySQLDialect
</property>
<property name = "hibernate.connection.driver_class">
com.mysql.jdbc.Driver
</property>
<!-- Assume Emp is the database name -->
<property name = "hibernate.connection.url">
jdbc:mysql://localhost:3308/Empdata
</property>
<property name = "hibernate.connection.username">
root
</property>
<property name = "hibernate.connection.password">
appu
</property>
<property name= "hbm2ddl.auto">update
</property>
</session-factory>
</hibernate-configuration>
```

Employee class

```
package Employee.Manage;
import javax.persistence.*;
@Entity
@Table(name = "EMPLOYEE")
```

```java
public class Employee {
@Id @GeneratedValue
@Column(name = "id")
private int id;
@Column(name = "first_name")
private String firstName;
@Column(name = "last_name")
private String lastName;
@Column(name = "salary")
private int salary;
public Employee() {}
public int getId() {
return id;
}
public void setId( int id ) {
this.id = id;
}
public String getFirstName() {
return firstName;
}
public void setFirstName( String first_name ) {
this.firstName = first_name;
}
public String getLastName() {
return lastName;
}
public void setLastName( String last_name ) {
this.lastName = last_name;
}
public int getSalary() {
return salary;
}
public void setSalary( int salary ) {
this.salary = salary;
}}
```

Main Class

```
package Employee.Manage;
import java.util.List;
import java.util.Date;
import java.util.Iterator;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
@SuppressWarnings("deprecation")
public class ManageEmployee {
private static SessionFactory factory;
public static void main(String[] args) {
try {
factory = new AnnotationConfiguration().
configure().
//addPackage("com.xyz") //add package if used.
addAnnotatedClass(Employee.class).
buildSessionFactory();
} catch (Throwable ex) {
System.err.println("Failed to create sessionFactory object." + ex);
throw new ExceptionInInitializerError(ex);
}
ManageEmployee ME = new ManageEmployee();
/* Add few employee records in database */
Integer empID1 = ME.addEmployee("David","Austin", 10000);
Integer empID2 = ME.addEmployee(" Valli","Pataballa", 30000);
Integer empID3 = ME.addEmployee("Diana","Lorentz", 25000);
Integer empID4 = ME.addEmployee("Nancy","Greenberg", 35000);
/* List down all the employees */
ME.listEmployees();
/* Update employee's records */
```

```
ME.updateEmployee(empID1, 30000);
ME.updateEmployee(empID2, 45000);
ME.updateEmployee(empID3, 40000);
ME.updateEmployee(empID4, 50000);
/* Delete an employee from the database */
ME.deleteEmployee(empID1);
/*Add an employee from the database*/
ME.addEmployee("Peter","Vargas",20000 );
ME.addEmployee("Jack","Livingston",22000 );
/* List down new list of the employees */
ME.listEmployees();
}
/* Method to CREATE an employee in the database */
public Integer addEmployee(String fname, String lname, int salary){
Session session = factory.openSession();
Transaction tx = null;
Integer employeeID = null;
try {
tx = session.beginTransaction();
Employee employee = new Employee();
employee.setFirstName(fname);
employee.setLastName(lname);
employee.setSalary(salary);
employeeID = (Integer) session.save(employee);
tx.commit();
} catch (HibernateException e) {
if (tx!=null) tx.rollback();
e.printStackTrace();
} finally {
session.close();
}
return employeeID;
}
/* Method to READ all the employees */
public void listEmployees( ){
```

```
Session session = factory.openSession();
Transaction tx = null;
try {
tx = session.beginTransaction();
List employees = session.createQuery("FROM Employee").list();
for (Iterator iterator = employees.iterator(); iterator.hasNext();){
Employee employee = (Employee) iterator.next();
System.out.print("First Name: " + employee.getFirstName());
System.out.print(" Last Name: " + employee.getLastName());
System.out.println(" Salary: " + employee.getSalary());
}
tx.commit();
} catch (HibernateException e) {
if (tx!=null) tx.rollback();
e.printStackTrace();
} finally {
session.close();
}
}
/* Method to UPDATE salary for an employee */
public void updateEmployee(Integer EmployeeID, int salary ){
Session session = factory.openSession();
Transaction tx = null;
try {
tx = session.beginTransaction();
Employee employee = (Employee)session.get(Employee.class,
EmployeeID);
employee.setSalary( salary );
                session.update(employee);
tx.commit();
} catch (HibernateException e) {
if (tx!=null) tx.rollback();
e.printStackTrace();
} finally {
session.close();
```

```
}
}
/* Method to DELETE an employee from the records */
public void deleteEmployee(Integer EmployeeID){
Session session = factory.openSession();
Transaction tx = null;
try {
tx = session.beginTransaction();
Employee employee = (Employee)session.get(Employee.class,
EmployeeID);
session.delete(employee);
tx.commit();
} catch (HibernateException e) {
if (tx!=null) tx.rollback();
e.printStackTrace();
} finally {
session.close();
}
}
public Integer addEmployee1(String string, String string2, int i) {
		// TODO Auto-generated method stub
		return null;
	}
public void deleteEmployee1(Integer empID2) {
		// TODO Auto-generated method stub
	}
	public void updateEmployee1(Integer empID1, int i) {
		// TODO Auto-generated method stub
	}
	public void listEmployees1() {
		// TODO Auto-generated method stub
	}}
```

Employee table in sql

```
mysql> select * from employee;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 20 | David      | Austin    |  10000 |
| 21 |  Valli     | Pataballa |  30000 |
| 22 | Diana      | Lorentz   |  25000 |
| 23 | Nancy      | Greenberg |  35000 |
+----+------------+-----------+--------+
4 rows in set (0.00 sec)

mysql>
```

Employee update salary

```
mysql> select * from employee;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 20 | David      | Austin    |  10000 |
| 21 |  Valli     | Pataballa |  30000 |
| 22 | Diana      | Lorentz   |  25000 |
| 23 | Nancy      | Greenberg |  35000 |
| 24 | David      | Austin    |  30000 |
| 25 |  Valli     | Pataballa |  45000 |
| 26 | Diana      | Lorentz   |  40000 |
| 27 | Nancy      | Greenberg |  50000 |
+----+------------+-----------+--------+
8 rows in set (0.00 sec)

mysql>
```

Delete Employee

```
mysql> select * from employee;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
| 20 | David      | Austin    |  10000 |
| 21 |  Valli     | Pataballa |  30000 |
| 22 | Diana      | Lorentz   |  25000 |
| 23 | Nancy      | Greenberg |  35000 |
| 24 | David      | Austin    |  30000 |
| 25 |  Valli     | Pataballa |  45000 |
| 26 | Diana      | Lorentz   |  40000 |
| 27 | Nancy      | Greenberg |  50000 |
| 29 |  Valli     | Pataballa |  45000 |
| 30 | Diana      | Lorentz   |  40000 |
| 31 | Nancy      | Greenberg |  50000 |
+----+------------+-----------+--------+
11 rows in set (0.00 sec)
```

Add Employee

```
mysql> select * from employee;
+----+------------+------------+--------+
| id | first_name | last_name  | salary |
+----+------------+------------+--------+
| 20 | David      | Austin     |  10000 |
| 21 |  Valli     | Pataballa  |  30000 |
| 22 | Diana      | Lorentz    |  25000 |
| 23 | Nancy      | Greenberg  |  35000 |
| 24 | David      | Austin     |  30000 |
| 25 |  Valli     | Pataballa  |  45000 |
| 26 | Diana      | Lorentz    |  40000 |
| 27 | Nancy      | Greenberg  |  50000 |
| 29 |  Valli     | Pataballa  |  45000 |
| 30 | Diana      | Lorentz    |  40000 |
| 31 | Nancy      | Greenberg  |  50000 |
| 33 |  Valli     | Pataballa  |  45000 |
| 34 | Diana      | Lorentz    |  40000 |
| 35 | Nancy      | Greenberg  |  50000 |
| 36 | Peter      | Vargas     |  20000 |
| 37 | Jack       | Livingston |  22000 |
+----+------------+------------+--------+
16 rows in set (0.00 sec)
```

2.Write and explain hibernate.cfg and hibernate.hbm file usage in ORM.

Hibernate configuration file:

As Hibernate can operate in different environments, it requires a wide range of configuration parameters. These configurations contain the mapping information that provides different functionalities to Java classes. Generally, we provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations either in an XML file (like hibernate.cfg.xml) or properties file (like hibernate.properties).

An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable SessionFactory.

We can acquire the Configuration class instance by instantiating it directly and specifying mappings in the configuration file. Use the addResource() method, if the mapping files are present in the classpath.

Configuration cfg = new configuration()

.addResource(“employee.hbm.xml”):

XML Based configuration:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
4.         "http://www.hibernate.org/dtd/hibernate-configuration-5.3.dtd">
5. <hibernate-configuration>
6.     <session-factory>
```

```
7.      <property name="hbm2ddl.auto">update</property>
8.    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
9.  <propertyname="connection.url">jdbc:oracle:thin:@localhost:1521:xe</pro
    perty>
10.           <property name="connection.username">system</property>
11.        <property name="connection.password">jtp</property>
12.     <propertyname="connection.driver_class">oracle.jdbc.driver.OracleDriv
    er</property>
13.         </session-factory>
14.     </hibernate-configuration>
```

Hibernate Configuration Properties:

Property	Description
hibernate.connection.driver_class	It represents the JDBC driver class.
hibernate.connection.url	It represents the JDBC URL.
hibernate.connection.username	It represents the database username.
hibernate.connection.password	It represents the database password.

Hibernate.connection.pool_size	**It represents the maximum number of connections available in the connection pool.**
hibernate.dialect	**It represents the type of database used in hibernate to generate SQL statements for a particular relational database.**

Hibernate Mapping file:

The full name of HBM is Hibernate Mapping. It is an XML file in which we define the mapping between POJO class to the database table and POJO class variables to table columns. The resource file hibernate.cfg.xml, which supports to represent the Hibernate configuration information. The conection.driver_class, connection.URL, connection.username, and connection.password property element that characterizes the JDBC connection information. The connection.pool_size is used to configure Hibernate's built-in connection pool how many connections to the pool. The Hibernate XML mapping file which includes the mapping correlation between the Java class and the database table. It is mostly named "xx.hbm.xml" and represents in the Hibernate configuration file "hibernate.cfg.xml."

XML Mapping File:

```
<hibernate-configuration>

<session-factory>

  <property
name="hibernate.bytecode.use_reflection_optimizer">false</property>

  <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>

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

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

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

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

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

  <mapping resource="com/demo/common/HiberDemo.hbm.xml"></mapping>

</session-factory>

</hibernate-configuration>
```

Usage of the Mapping file:

- **Ø The mapping document is an XML document having <hibernate-mapping> as the root element, which contains all the <class> elements.**
- **Ø The <class> elements are used to define specific mappings from a 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.**
- **Ø The <meta> element is optional element and can be used to create the class description.**

3. Explain advantages of HQL and Caching in Hibernate.

Advantages of HQL:

1.Open source and lightweight.

Hibernate framework is open source under the LGPL license and lightweight.

2.Fast Performance.

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3.Database independent Query.

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4.Automatic table creation.

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5.Simplifies complex joins.

Fetching data from multiple tables is easy in hibernate framework.

6.Provide query statistics and Database status.

Hibernate supports Query cache and provide statistics about query and database status.

Caching in Hibernate:

Hibernate caching improves the performance of the application by pooling the object in the cache. It is useful when we have to fetch the same data multiple times.

There are mainly two types of caching.

- **First level caching.**
- **Second level caching.**

First level caching:

Session object holds the first level cache data. It is enabled by default. The first level cache data will not be available to entire application. An application can use many session object.

Second level caching:

SessionFactory object holds the second level cache data. The data stored in the second level cache will be available to entire application. But we need to enable it explicitely.

- **EH (Easy Hibernate) Cache**
- **Swarm Cache**
- **OS Cache**
- **JBoss Cache.**

Advantages of Hibernate Caching

After learning "What is Hibernate Caching," Now, let's discuss some advantages of Hibernate Caching:

Hibernate is better than JDBC.

Mapping of Domain object to the relational database.

It supports Layered Architecture, and you can use the components as per application requirements.

It supports JPA and can work as a JPA provider.

It supports the Standard ORM solution.

It is Database Independent.

4.Describe Sessionfactory, Session, Transaction of objects.

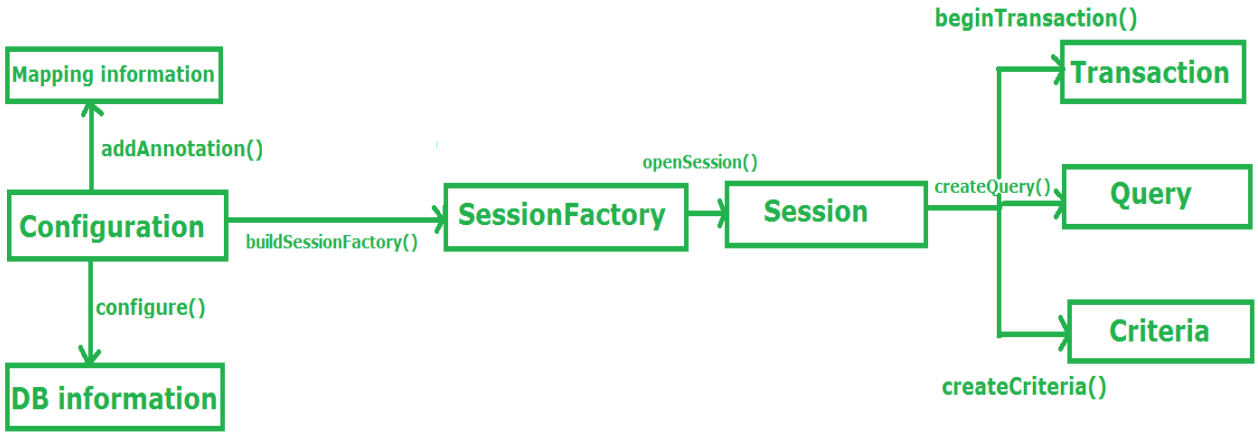

Fig: Hibernate Architecture

Session factory:

SessionFactory is an interface. SessionFactory can be created by providing Configuration object, which will contain all DB related property details pulled from either hibernate.cfg.xml file or hibernate.properties file. SessionFactory is a factory for Session objects.
We can create one SessionFactory implementation per database in any application. If your application is referring to multiple databases, then you need to create one SessionFactory per database.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. The SessionFactory is a thread safe object and used by all the threads of an application.

Session:

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 them as needed. The main function of the Session is to offer, create, read, and delete operations for instances of mapped entity classes.

Transaction in object:

A transaction is associated with a Session and is usually instantiated by a call to Session. beginTransaction(). A single session might span multiple transactions However, it is intended that there will be at most one uncommitted Transaction associated with a particular Session at any time.