1. **Hibernate CRUD Operations**

   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.product</groupId>
  <artifactId>Hybernate</artifactId>
  <version>0.0.1-SNAPSHOT</version>


  <dependencies>

 <!-- Hibernate 4.3.6 Final -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>4.3.6.Final</version>
</dependency>
<!-- Mysql Connector -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.21</version>
</dependency>
</dependencies>
</project>
```

Hybernate.cfg.xml

```
<?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>
<property
name="hibernate.bytecode.use_reflection_optimizer">false</pr
operty>
<property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driv
er</property>
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/
product</property>
<property
name="hibernate.connection.username">root</property>
<property
name="hibernate.connection.password">root</property>
<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect<
/property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
</session-factory>
</hibernate-configuration>
```

Employee Class

```
package Employee.manage;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
     @Entity
     @Table(name = "EMPLOYEE")
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 product.process;
import java.util.List;
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;

@SuppressWarnings("deprecation")
public class Manage_Employee {

 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);
            }

            Manage_Employee ME = new Manage_Employee();


            /* Add few employee records in database */
```

```
            Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
            Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
            Integer empID3 = ME.addEmployee("John", "Paul", 10000);

            /* List down all the employees */
            ME.listEmployees();

            /* Update employee's records */
            ME.updateEmployee(empID1, 5000);

            /* Delete an employee from the database */
            ME.deleteEmployee(empID2);

            /* 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();
            }
         }
      }
```

OUTPUT:

Employee table in sql

```
mysql> CREATE TABLE Employee (
    ->     id INT PRIMARY KEY AUTO_INCREMENT,
    ->     first_name VARCHAR(50),
    ->     last_name VARCHAR(50),
    ->     salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO Employee (first_name, last_name, salary)
    -> VALUES
    ->     ("Zara", "Ali", 1000),
   ->     ("Daisy", "Das", 5000),
```

```
    ->     ("John", "Paul", 10000);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Employee;
+----+------------+-----------+----------+
| id | first_name | last_name | salary   |
+----+------------+-----------+----------+
|  1 | Zara       | Ali       |  1000.00 |
|  2 | Daisy      | Das       |  5000.00 |
|  3 | John       | Paul      | 10000.00 |
+----+------------+-----------+----------+
3 rows in set (0.00 sec)

mysql> UPDATE Employee
    -> SET salary = 6000
    -> WHERE first_name = "Zara" AND last_name = "Ali";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> DELETE FROM Employee
    -> WHERE first_name = "John" AND last_name = "Paul";
Query OK, 1 row affected (0.00 sec)
```

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

Hibernate Configuration

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")
```

Let's see an example to provide the configurations in XML file and properties file.

XML Based Configuration

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

   <property name="hbm2ddl.auto">update</property>
  <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
  <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</propert
y>
 <property name="connection.username">system</property>
      <property name="connection.password">jtp</property>
  <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</pro
perty>
      </session-factory>
</hibernate-configuration>
```

Properties of Hibernate Configuration

Hibernate Configuration Properties

Property	Description
hibernate.dialect	It represents the type of database used in hibernate to generate SQL statements for a particular relational database.
hibernate.show_sql	It is used to display the executed SQL statements to console.
hibernate.format_sql	It is used to print the SQL in the log and console.
hibernate.default_catalog	It qualifies unqualified table names with the given catalog in generated SQL.
hibernate.default_schema	It qualifies unqualified table names with the given schema in generated SQL.
hibernate.session_factory_name	The SessionFactory interface automatically bound to this name in JNDI after it has been created.
hibernate.default_entity_mode	It sets a default mode for entity representation for all sessions opened from this SessionFactory
hibernate.order_updates	It orders SQL updates on the basis of the updated primary key.
hibernate.use_identifier_rollback	If enabled, the generated identifier properties will be reset to default values when objects are deleted.
hibernate.generate_statistics	If enabled, the Hibernate will collect statistics useful for performance tuning.

hibernate.use_sql_comments	If enabled, the Hibernate generate comments inside the SQL. It is used to make debugging easier.

Hibernate Cache Properties

Property	Description
hibernate.cache.provider_class	It represents the classname of a custom CacheProvider.
hibernate.cache.use_minimal_puts	It is used to optimize the second-level cache. It minimizes writes, at the cost of more frequent reads.
hibernate.cache.use_query_cache	It is used to enable the query cache.
hibernate.cache.use_second_level_cache	It is used to disable the second-level cache, which is enabled by default for classes which specify a mapping.
hibernate.cache.query_cache_factory	It represents the classname of a custom QueryCache interface.
hibernate.cache.region_prefix	It specifies the prefix which is used for second-level cache region names.
hibernate.cache.use_structured_entries	It facilitates Hibernate to store data in the second-level cache in a more human-friendly format.

Hibernate Transaction Properties

Property	Description
hibernate.transaction.factory_class	It represents the classname of a TransactionFactory which is used with Hibernate Transaction API.
hibernate.transaction.manager_lookup_class	It represents the classname of a TransactionManagerLookup. It is required when JVM-level caching is enabled.
hibernate.transaction.flush_before_completion	If it is enabled, the session will be automatically flushed during the before completion phase of the transaction.
hibernate.transaction.auto_close_session	If it is enabled, the session will be automatically closed during the after completion phase of the transaction.

Hibernate Mapping file

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

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"

<hibernate-mapping>

   <class name="com.example.Employee" table="Employee">

      <id name="id" type="int">

         <column name="employee_id" />

         <generator class="native" />

      </id>

      <property name="firstName" type="string">

         <column name="first_name" />

      </property>

      <property name="lastName" type="string">

         <column name="last_name" />

      </property>

      <property name="salary" type="big_decimal">

         <column name="salary" />

      </property>

   </class>
```

</hibernate-mapping>

Usage of the Mapping file:

The mapping document is an XML document having as the root element, which contains all the elements. The 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. The element is optional element and can be used to create the class description.

3.Explain advantages of HQL and Caching in Hibernate.

Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

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 Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides 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.

1 First level caching.

2 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.

Ø EH (Easy Hibernate) Cache

Ø Swarm Cache

Ø OS Cache

Ø JBoss Cache.

ADVANTAGES OF HYBERNATE 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 Session factory, Session, Transaction of objects.

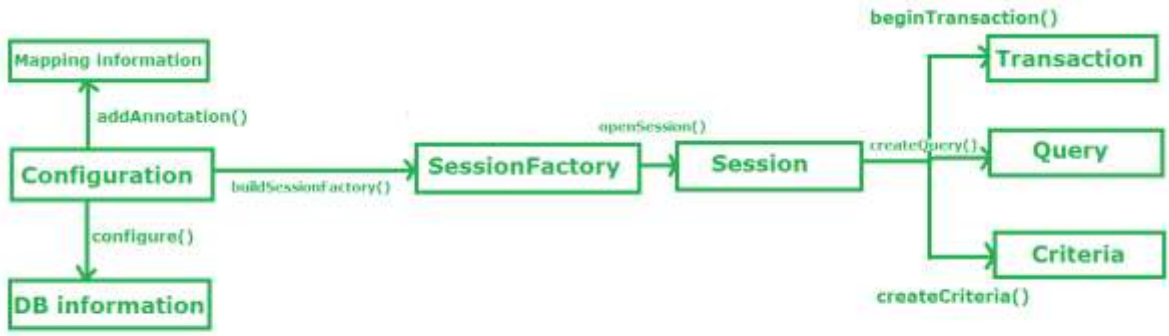

Fig: Hibernate Architecture

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.

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 OBJECT:

- 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();