Q1. What is Hibernate? List the advantages of hibernate over JDBC.

Ans.

- Hibernate is used convert object data in JAVA to relational database tables.
- It is an open source Object-Relational Mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

JDBC	Hibernate
JDBC maps Java classes to database tables (and from Java data types to SQL data types)	Hibernate automatically generates the queries.
With JDBC, developer has to write code to map an object model's data to a relational data model.	Hibernate is flexible and powerful ORM to map Java classes to database tables.
With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java. So with JDBC, mapping between Java objects and database tables is done manually.	Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
Require JDBC Driver for different types of database.	Makes an application portable to all SQL databases.
Handles all create-read-update-delete (CRUD) operations using SQL Queries.	Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
Working with both Object-Oriented software and Relational Database is complicated task with JDBC.	Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
JDBC supports only native Structured Query Language (SQL)	Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database)

List the advantages of hibernate over JDBC

- Hibernate is flexible and powerful ORM to map Java classes to database tables.
- Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
- Hibernate automatically generates the queries.
- It makes an application portable to all SQL databases.
- Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
- Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
- Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database).
- Hibernate supports Inheritance, Associations, Collections.

- Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One.
- Hibernate provided Dialect classes, so we no need to write SQL queries in hibernate, instead we use the methods provided by that API.

Q2. Draw and explain the architecture of Hibernate.

Ans.

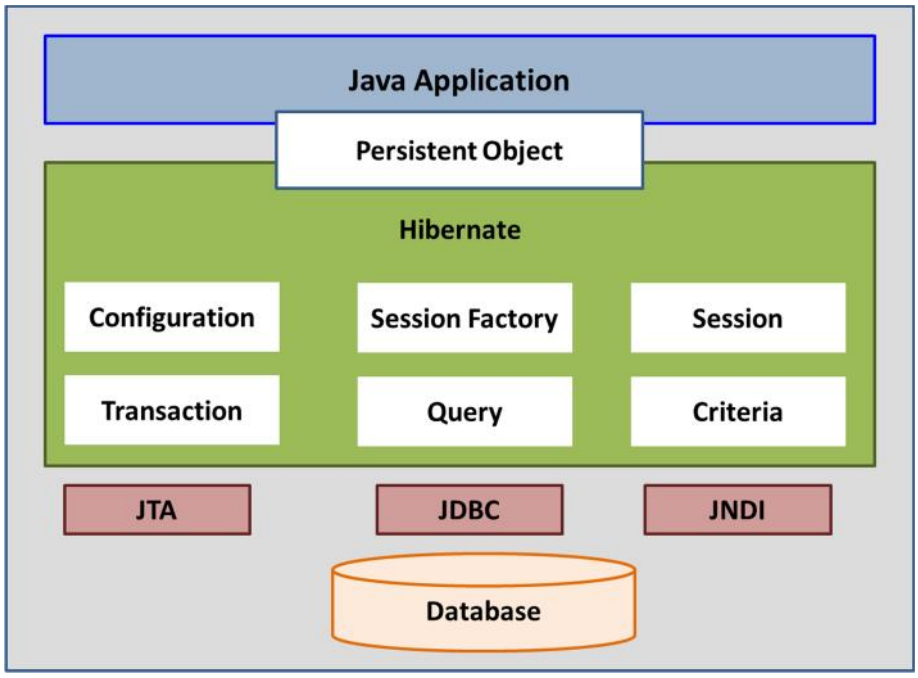


Figure: Hibernate Architecture

- For creating the first hibernate application, we must know the objects/elements of Hibernate architecture.
- They are as follows:
  i. Configuration
  ii. Session factory
  iii. Session
  iv. Transaction factory
  v. Query
  vi. Criteria

1. **Configuration Object**
   - The Configuration object is the first Hibernate object you create in any Hibernate application.
   - It is usually created only once during application initialization.
   - 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**.

- **Class Mapping Setup**:
  - This component creates the connection between the Java classes and database tables.

2. SessionFactory Object

- The SessionFactory is a thread safe object and used by all the threads of an application.
- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application.
- 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.

3. 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.
- 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.

4. 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).

5. 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.

6. Criteria Object

- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Q3. What is HQL? How does it different from SQL? List its advantages.

Ans. ***What is HQL?***

- The Hibernate ORM framework provides its own query language called Hibernate Query Language.
- 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.
- Therefore, it is database independent query language.

How does it different from SQL?

SQL	HQL
SQL is based on a relational database model	HQL is a combination of object-oriented programming with relational database concepts.
SQL manipulates data stored in tables and modifies its rows and columns.	HQL is concerned about objects and its properties.
SQL is concerned about the relationship that exists between two tables.	HQL considers the relation between two objects.

Advantages of HQL:

- Provides full support for relation operations
- Returns results as objects
- Support polymorphic queries
- Easy to learn and use
- Supports for advanced features
- Provides database independency

Q4. What is O/R Mapping? How it is implemented using Hibernate. Give an example of Hibernate XML mapping file.

Ans. Three most important mapping are as follows:

1. Collections Mappings
2. Association Mappings
3. Component Mappings

Collections Mappings

- If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.
- Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

Association Mappings:

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.
- There are the four ways in which the cardinality of the relationship between the objects can be expressed.
- An association mapping can be unidirectional as well as bidirectional.

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

Component Mappings:

If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class.
The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

Give an example of Hibernate XML mapping file.

```
<hibernate-mapping>
  <class name="hibernatetest.Customer" table="customers">
  <id column="C_ID" name="customerID" type="int">
  <generator class="native">
  </generator></id>
  <property name="customerName">
  <column name="name">
  </column></property>
  <property name="customerAddress">
  <column name="address">
  </column></property>
  <property name="customerEmail">
  <column name="email">
  </column></property>
  </class>
</hibernate-mapping>
```

Q5. Explain the Hibernate cache architecture.

Ans.

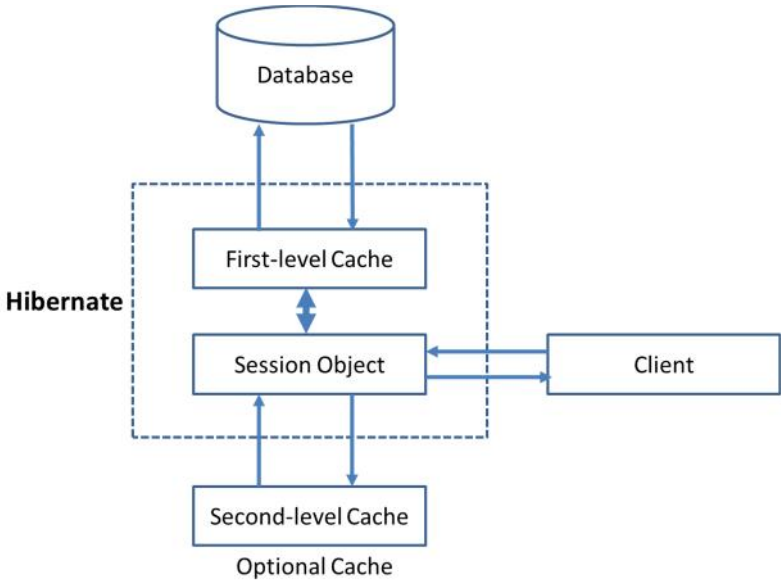


Figure: Hibernate Cache Architecture

- Caching is all about application performance optimization.
- It is situated between your application and the database to avoid the number of database hits as many as possible.
- To give a better performance for critical applications.

First-level cache:

- The first-level cache is the Session cache.
- The Session object keeps an object under its own control before committing it to the database.
- If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued.
- If you close the session, all the objects being cached are lost.

Second-level cache:

- It is responsible for caching objects across sessions.
- Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache.
- Any third-party cache can be used with Hibernate.
- An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.
- While preparing a Hibernate mapping document, we map the Java data types into RDBMS data types.
- The types declared and used in the mapping files are not Java data types; they are not SQL database types either.
- These types are called ***Hibernate mapping types***, which can translate from Java to SQL data types and vice versa.

Q6. Write a program to insert record in to the database using hibernate.

Ans. **Steps to run first hibernate example with MySQL in Netbeans IDE 8.2**

Step-1: Create the database

```
CREATE DATABASE retailer;
```

Step-2: Create table result

```
CREATE TABLE customers(
     name varchar(20),
     C_ID int NOT NULL AUTO_INCREMENT,
     address  varchar(20),
     email varchar(50),
     PRIMARY KEY(C_ID)
);
```

Step-3: Create new java application.

```
File > New project > Java > Java Application > Next
Name it as HibernateTest.
Then click Finish to create the project.
```

Step-4: Create a POJO(Plain Old Java Objects) class

- We create this class to use variables to map with the database columns.
- Right click the package (hibernatetest) & select New > Java Class Name it as Customer.
- Click Finish to create the class.

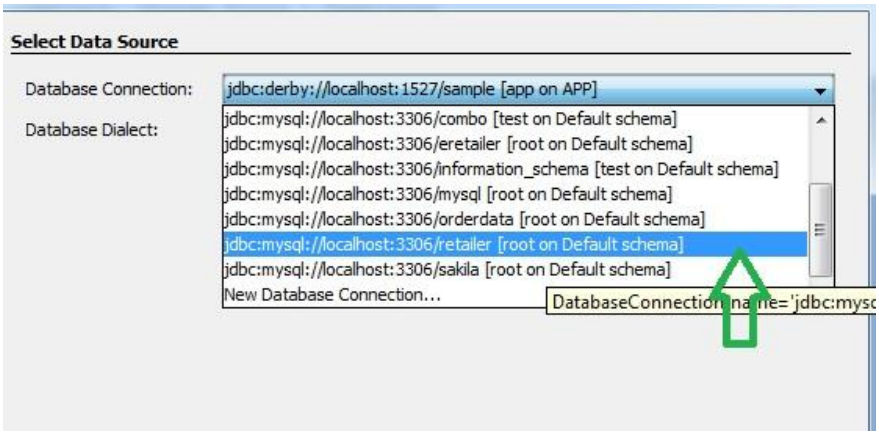
```
package hibernatetest;
public class Customer {
    private String customerName;
    private int customerID;
    private String customerAddress;
private String customerEmail;
public void setCustomerAddress(String customerAddress) {
         this.customerAddress = customerAddress;}
public void setCustomerEmail(String customerEmail) {
      this.customerEmail = customerEmail;}
public void setCustomerID(int customerID) {
      this.customerID = customerID; }
public void setCustomerName(String customerName) {
        this.customerName = customerName;    }
public String getCustomerAddress() {
        return customerAddress;    }
public String getCustomerEmail() {
        return customerEmail;    }
public int getCustomerID() {
        return customerID;    }
public String getCustomerName() {
        return customerName;    }}
```

Step-5: Connect to the database we have already created. [retailer]

- Select Services tab lying next to the Projects tab.
- Expand Databases.
- Expand MySQL Server. There we can see the all databases on MySQL sever
- Right click the database retailer. Select Connect.

Step-6: Creating the configuration XML

- Hibernate need a configuration file to create the connection.
- Right click package hibernatetest select New > Other > Hibernate > Hibernate Configuration Wizard.
- Click Next > In next window click the drop down menu of Database Connection and select retailer database connection.



hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
          <property name="hibernate.connection.driver_class">
                              com.mysql.jdbc.Driver </property>
          <property name="hibernate.connection.url">

           jdbc:mysql://localhost:3306/retailer</property>
          <property name="hibernate.connection.username">
                                    root</property>
          <property name="hibernate.connection.password">
                                    root</property>
          <property name="hibernate.connection.pool_size">
                                      10</property>
          <property name="hibernate.dialect">
                     org.hibernate.dialect.MySQLDialect</property>
         <property name="current_session_context_class">
                                  thread</property>
```

```
        <property name="cache.provider_class">

        org.hibernate.cache.NoCacheProvider</property>
        <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">
                          update</property>

        <mapping resource="hibernate.hbm.xml"></mapping>
     </session-factory>
</hibernate-configuration>
```

Step-7: Creating the mapping file [hibernate.hbm]

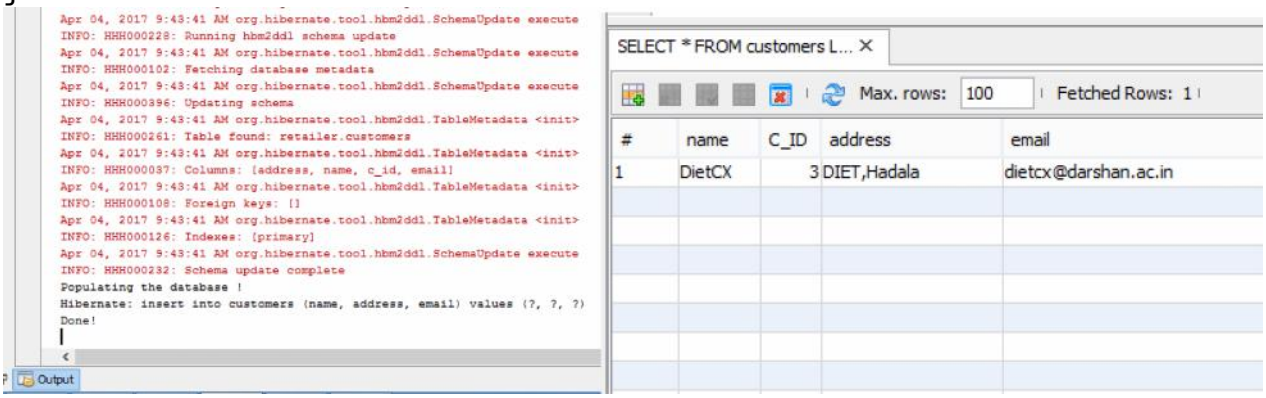
- Mapping file will map relevant java object with relevant database table column.
- Right click project select New > Other > Hibernate > Hibernate Mapping Wizard
- click Next name it as hibernate.hbm
- click Next> In next window we have to select Class to Map and Database Table.
- After selecting correct class click OK
  Select Database Table
  Click drop down list and select the table you want to map.
  Code for mapping file.

hibernate.hbm.xml

```
<hibernate-mapping>
  <class name="hibernatetest.Customer" table="customers">
       <id column="C_ID" name="customerID" type="int">
       <generator class="native">    </generator></id>
       <property name="customerName">
            <column name="name">  </column>
       </property>
       <property name="customerAddress">
            <column name="address">  </column>
        </property>
       <property name="customerEmail">
            <column name="email">  </column>
       </property>
  </class>
</hibernate-mapping>
```

Step-8: Now java program to insert record into the database

```
package hibernatetest;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class HibernateTest {
  public static void main(String[] args) {
  Session session = null;
  try
  {
     SessionFactory sessionFactory = new
org.hibernate.cfg.Configuration().configure().buildSessionFactory(
);
     session =sessionFactory.openSession();
        session.beginTransaction();
        System.out.println("Populating the database !");
        Customer customer = new Customer();
        customer.setCustomerName("DietCX");
        customer.setCustomerAddress("DIET,Hadala");
        customer.setCustomerEmail("dietcx@darshan.ac.in");
        session.save(customer);
        session.getTransaction().commit();
        System.out.println("Done!");
        session.flush();
        session.close();
     }catch(Exception ){System.out.println(e.getMessage());    } }
}
```



Q7. Develop program to get all students data from database using hibernate. Write necessary xml files.

Ans.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class HibSelect {
 public static void main(String[] args) {
  Session session = null;
     try
     {
     SessionFactory sessionFactory = new
org.hibernate.cfg.Configuration().configure().buildSessionFactory(
);
   session =sessionFactory.openSession();
   session.beginTransaction();
   System.out.println("Populating the database !");

   hibernatetest.Customer customer= new hibernatetest.Customer();
   Query query=session.createQuery("from hibernatetest.Customer");
                          //here persistent class name is Emp
   List list=query.list();
   while(list.isEmpty())
   {
      System.out.println(list.get(0).toString());
      System.out.println(list.get(1).toString());
      System.out.println(list.get(2).toString());
   }

session.save(customer);

        session.getTransaction().commit();
        System.out.println("Done!");
        session.flush();
        session.close();
     }catch(Exception e){System.out.println(e.toString());
     } } }
```

Write necessary xml files.

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
          <property name="hibernate.connection.driver_class">
                              com.mysql.jdbc.Driver </property>
          <property name="hibernate.connection.url">

           jdbc:mysql://localhost:3306/retailer</property>
          <property name="hibernate.connection.username">
                                    root</property>
          <property name="hibernate.connection.password">
                                    root</property>
          <property name="hibernate.connection.pool_size">
                                      10</property>
          <property name="hibernate.dialect">
                     org.hibernate.dialect.MySQLDialect</property>
         <property name="current_session_context_class">
                                  thread</property>
         <property name="cache.provider_class">

         org.hibernate.cache.NoCacheProvider</property>
         <property name="show_sql">true</property>
         <property name="hibernate.hbm2ddl.auto">
                             update</property>

         <mapping resource="hibernate.hbm.xml"></mapping>
      </session-factory>
</hibernate-configuration>
```

hibernate.hbm.xml

```
   <hibernate-mapping>
     <class name="hibernatetest.Customer" table="customers">
          <id column="C_ID" name="customerID" type="int">
          <generator class="native">    </generator></id>
          <property name="customerName">
               <column name="name">  </column>
          </property>
          <property name="customerAddress">
               <column name="address">  </column>
           </property>
          <property name="customerEmail">
               <column name="email">  </column>
          </property>
     </class>
   </hibernate-mapping>
```

Q8. Explain Hibernate Annotation.

Ans.

- Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping.
- Consider we are going to use following EMPLOYEE table to store our objects:

```
create table EMPLOYEE (
id INT NOT NULL auto_increment,
first_name VARCHAR(20) default NULL,
 last_name VARCHAR(20) default NULL,
salary INT default NULL,
PRIMARY KEY (id) );
```

- Following is the mapping of Employee class with annotations to map objects with the defined EMPLOYEE table:

```
import javax.persistence.*;
@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;
}
}
```

- @Entity Annotation:
  - Employee class which marks this class as an entity bean
- @Table Annotation:
  - The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.
- @Id and @GeneratedValue Annotations:
  - Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation

  @GeneratedValue is same as Auto Increment.
- @Column Annotation:
  - The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:
  - **name** attribute permits the name of the column to be explicitly specified.
  - **length** attribute permits the size of the column used to map a value particularly for a String value.
  - **nullable** attribute permits the column to be marked NOT NULL when the schema is generated.
  - **unique** attribute permits the column to be marked as containing only unique values.