Hibernate –

Q1. Do Hibernate Configuration and Create a table in the database using DAO (Data Access Object) class having fields such as id, name, and address.

-> Configurations -

1. Go to File new Project -> Maven Project -> next -> web app archetype -> Project name -> Finish

2. come to pom.xml and add dependency <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.15.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>
```

3. Go to the Project name and right-click to build path and add JRE System Library, Maven Dependency, and Server Runtime from Add Library.

4. Create hibernate.cfg.xml file - src/main/java/new/other/xml file / hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
```

```
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/dabase_name?
            useSSL=False</property>
        <property name="connection.username">username</property>
        <property name="connection.password">password</property>
        <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <property name="hbm2ddl.auto">create</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>

            // Do mapping for Servlet classes
        <mapping class="com.servlet.Add"/>


        <mapping class="com.dao.User"/>



        <mapping class="com.provider.FactoryProvider"/>

      </session-factory>

    </hibernate-configuration>
```

5. Right-click on src/main/java and create a package (i.e. com.example)
6. Right-click on the package to create a Java class as Customer
7. Do Servlet mapping in hibernate. cfg.xml file <mapping class="com.example.Customer"/>

Code :

```
    package com.example;
```

```
import javax.persistence.*;

@Entity
@Table(name="customer")
public class Customer {

	@Id
	int id;
	String name;
	String address;

	// Parent Constructor
	public Customer() {
		super();
		// TODO Auto-generated constructor stub
	}

	// Constructor of class
	public Customer(int id, String name, String address) {
		super();
		this.id = id;
		this.name = name;
		this.address = address;
	}


	// Getters and Setters
	public int getId() {
		return id;
	}
```

```
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}



}
```

Q2. Do Insert Operation in the above table as (id, name, address) as (1, ram, Latur), (2, shyam, Kolhapur). Further, Display the name from the given id.

-> 1. Create Servlet in Package - Add.java

```
	package com.servlet;


import java.io.IOException;


import org.hibernate.*;
import org.hibernate.cfg.Configuration;


import com.example.Customer;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;



@WebServlet("/add")
public class Add extends HttpServlet {
	private static final long serialVersionUID = 1L;

  public Add() {
    // TODO Auto-generated constructor stub
  }

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {



		Customer c1 = new Customer(1,"Ram","Latur");
		Customer c2 = new Customer(2,"Shyam","Kolphapur");


		// Prepare Session Factory Object


		SessionFactory sf = new Configuration().configure().buildSessionFactory();


		Session s = sf.openSession();


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

```
            s.save(c1);
            s.save(c2);

            Customer c = s.get(Customer.class, 1);

            PrintWriter out = response.getWriter();

            out.println(c.getName());

            tx.commit();
            s.close();
        }

}
```

2. Configure in hibernate.cfg.xml

3. Go to src/main/web app/ index.jsp

```
        <html>
        <body>
            <h2>Hello World!</h2>

            <form action="add" method="post">

                <input type="submit" value="Submit">
            </form>
        </body>
        </html>
```

4. Run Project

Q3. Write any Hibernate Program and work on different Annotations

1. @Entity 2. @Table 3. @Id 4. @GeneratedValue 5. @Column 6. @OneToMany 7. @ManyToOne 8. @Transient

-> Author.java

```
package com.example;

import java.util.*;

import javax.persistence.*;

@Entity

@Table(name="authors")
public class Author {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "author_id")
    private Long id;

    @Column(name = "author_name")
    private String name;

    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
    private List<Book> books;
```

```
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 List<Book> getBooks() {
		return books;
	}
```

```
	public void setBooks(List<Book> books) {
		this.books = books;
	}



	public Author(Long id, String name, List<Book> books) {
		super();
		this.id = id;
		this.name = name;
		this.books = books;
	}



	public Author() {
		super();
		// TODO Auto-generated constructor stub
	}



	// Add a book to the author's books list
    public void addBook(Book book) {
        if (books == null) {
            books = new ArrayList<Book>();
        }
        books.add(book);
        book.setAuthor(this);
    }
}
```

Book.java

```
package com.example;



import java.util.List;

import javax.persistence.*;

@Entity
@Table(name="book")
public class Book {


	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;

	@Column(name = "book_name")
	private String bookName;

	@ManyToOne
	@JoinColumn(name = "author_id")
	private Author author;

		public Book() {
			super();
			// TODO Auto-generated constructor stub
		}
```

```
public Book(Long id, String bookName, Author author) {
        super();
        this.id = id;
        this.bookName = bookName;
        this.author = author;
}

public Long getId() {
        return id;
}

public void setId(Long id) {
        this.id = id;
}

public String getBookName() {
        return bookName;
}

public void setBookName(String bookName) {
        this.bookName = bookName;
}

public Author getAuthor() {
        return author;
}

public void setAuthor(Author author) {
        this.author = author;
}
```

```
        }
```

Add.java

```
        Author author = new Author();
    author.setName("John");

    // Create some books
    Book book1 = new Book();
    book1.setBookName("Book 1");
    author.addBook(book1);

    Book book2 = new Book();
    book2.setBookName("Book 2");
    author.addBook(book2);

        // save to database
        s.save(author);
        s.save(book1);
        s.save(book2);
```

Q4. Write code for Hibernate Fetching Techniques with Lazy and Eager types in above code.

-> 1. Lazy

```
          @OneToMany(mappedBy = "author", fetch=FetchType.LAZY)
           private List<Book> books;
```

2. Eager

```
          @OneToMany(mappedBy = "author", fetch=FetchType.EAGER)
           private List<Book> books;
```

Q5. Using HQL (Hibernate Query Language) implement

1. Display 2. Delete 3. Update Operation

-> index.html

```
<form action="add" method="post">

<input type="text" name="book" placeholder="Enter Book name: ">
<input type="text" name="name" placeholder="Enter Author name: ">

<input type="submit" value="Submit">
</form>
<br>
<h1>HQL Query Click Here</h1>
<form action="hql" method="post">

<input type="submit" value="Submit">

<%-- <input type="text" name="dname" placeholder="Enter Author id to Delete: ">
<input type="submit" value="Delete">--%>

<input type="text" name="uid" placeholder="Enter Author id to Updated ">
<input type="text" name="uname" placeholder="Enter Author name to Updated ">

<input type="submit" value="Update">
</form>
```

HQLQuery.java

```
package com.hql;
```

```
import java.io.*;
import java.util.*;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.NativeQuery;

import com.example.Author;
import com.example.Book;
import com.example.Customer;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hql")
public class HQLQuery extends HttpServlet {
	private static final long serialVersionUID = 1L;

   public HQLQuery() {
      // TODO Auto-generated constructor stub
   }


	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

		PrintWriter out = response.getWriter();
```

```
		SessionFactory sf = new Configuration().configure().buildSessionFactory();

		Session s = sf.openSession();

		Transaction tx = s.beginTransaction();



		// Select display
		List<Author> query = s.createQuery("FROM Author",Author.class).list();

		for(Author a: query) {
			out.println("Auther :"+a.getName());
		}

		// Delete

//		int idToDelete = Integer.parseInt( request.getParameter("dname"));
//		String q = "DELETE FROM Customer WHERE id=:custid";
//	Query<Customer> delete = s.createQuery(q);
//	delete.setParameter("custid", idToDelete);
//	int deletedCount = delete.executeUpdate();
//	tx.commit();
//	out.println("Deleted " + deletedCount + " Customer.");



	// Update
	int id = Integer.parseInt(request.getParameter("uid"));
	String name1 = request.getParameter("uname");

	String query1 = "Update Customer set name =: custname where id=:custid";
```

```
        Query<Customer> update = s.createQuery(query1);
        update.setParameter("custid", id);
        update.setParameter("custname", name1);
        int updatecount = update.executeUpdate();
        tx.commit();
        out.println("Updated "+updatecount);


            }

}
```

Q6. Implement Level 1 Cache and Level 2 Cache in hibernate

-> (level 1 cache is provided by hibernate default)

1. Add Dependency in pom.xml
    1. ehcache dependency
    2. hibernate ehcache dependency
2. configure hibernate.xml
3. Perform Level 2 Cache