Hibernate Framework

DAY 1:

What is Framework?

- A framework is a software architecture that provides structure, guidelines, and reusable components to facilitate the development of an application.
- A framework is like a washing machine. It provides a ready-made structure (the machine body), predefined settings (like wash modes), and built-in functions (like rinse, spin, dry), so you can wash clothes without having to build a washing process from scratch.

Framework vs. Library:

- A framework is like a structured platform that calls your code and controls the overall flow of the application — you build your code around its rules and structure. On the other hand, a library is a collection of useful methods or classes that you call whenever needed, giving you full control over how and when those functions are used. In short, with a framework, it calls you (inversion of control), but with a library, you call it.

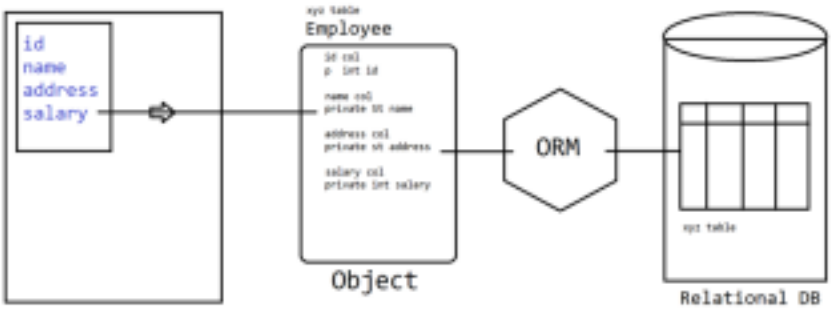
Why Framework?

- A framework is used in software development because it speeds up the process, ensures consistency, and reduces repetitive coding. It provides a predefined structure, reusable components, and built-in features like security, error handling, and database integration, allowing developers to focus on writing business logic instead of low-level setup. By following best practices enforced by the framework, code becomes more organized, maintainable, and scalable. Additionally, frameworks often come with strong community support and documentation, making problem-solving faster and development more efficient.

What is Hibernate?

- Hibernate is an Object-Relational Mapping (ORM) framework for Java that simplifies database interactions by mapping Java classes to database tables. Instead of writing complex SQL queries, you work

with Java objects and let Hibernate handle database operations behind the scenes.



Uses of Hibernate

- Simplifies CRUD (Create, Read, Update, Delete) operations in database programming.
- Manages database transactions and connection pooling.
- Supports complex object relationships like inheritance, associations, and collections.
- Provides database independence — easily switch databases without rewriting code.
- Handles lazy loading, caching, and query optimization automatically.
- Used widely in enterprise applications for data persistence.

When to Use Hibernate

- Complex Applications: When your application has complex object models with relationships (one-to-many, many-to-many) that need to be persisted.

- Database Independence: If you want your application to be easily portable across different databases without changing SQL queries.
- Rapid Development: To reduce boilerplate code and speed up development with automatic SQL generation and table creation.
- Caching Needs: When you want to leverage built-in caching to improve performance.
- Transaction Management: When you need simplified and consistent transaction handling.
- Maintenance & Scalability: For large-scale enterprise apps that require easier maintenance and scalability.

When Not to Use Hibernate

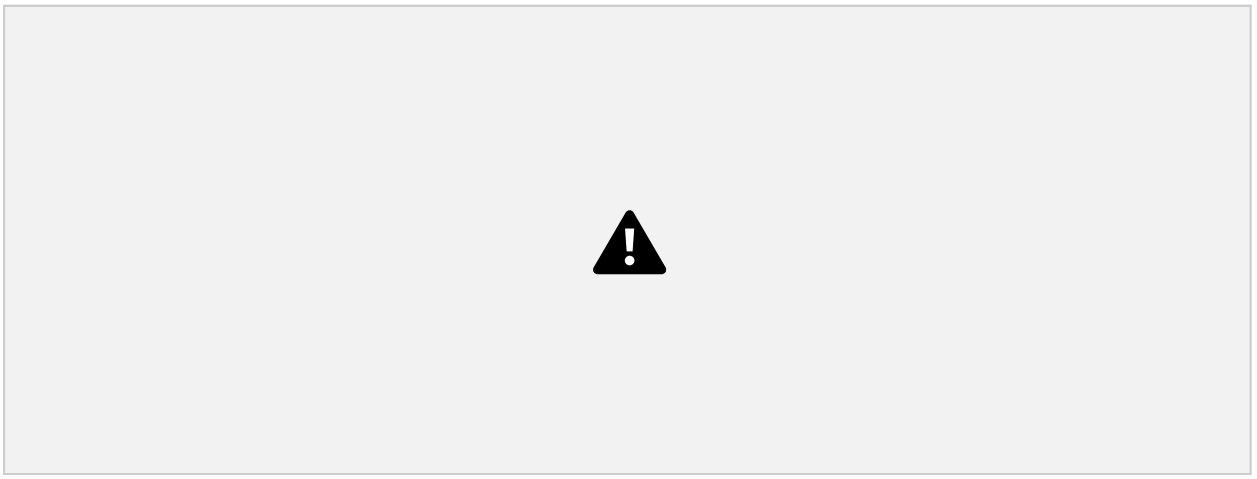
- Simple or Small Projects: If your application is very simple with minimal database interactions, plain JDBC might be sufficient.

Note: Hibernate is a JPA Implementation

- Hibernate is a popular implementation of the Java Persistence API (JPA) specification.
- While JPA defines the interfaces and rules for ORM in Java, it does not provide any working code by itself.
- Hibernate provides the actual code that implements these interfaces, enabling Java applications to perform database operations.
- Besides implementing JPA, Hibernate also offers additional features and optimizations beyond the JPA specification (NATIVE API).
- Therefore, to use JPA in a real project, you need a provider like Hibernate that implements the JPA interfaces such as EntityManager.

What is JPA?

- JPA (Java Persistence API) is a Java specification that standardizes the way Java objects are persisted to relational databases. It defines a set of interfaces and annotations to manage, store, update, and retrieve data between Java objects (entities) and database tables.
- JPA is NOT a framework or implementation by itself.
- It provides a standardized programming model for Object-Relational Mapping (ORM) in Java.



Hibernate as a JPA Implementation and Its Native API

- Hibernate is a popular implementation of the JPA specification. It implements standard JPA interfaces like EntityManager (from javax.persistence package), allowing you to write database code using JPA's standardized API.
- At the same time, Hibernate provides its own native API with classes like Session (from org.hibernate package) and Transaction for more advanced or Hibernate-specific features beyond JPA.
- for most applications, it's best to use the standard JPA API with Hibernate as the provider. This approach allows you to write portable and vendor-independent code using interfaces like EntityManager and EntityManagerFactory. By sticking to JPA standards, your application can easily switch to another JPA implementation if needed, and your code remains clean and maintainable. This is ideal for typical CRUD operations and most enterprise application requirements.
- However, if your project requires advanced Hibernate-specific features or finer control over ORM behavior, you should consider using Hibernate's native API. This involves working directly with Hibernate's Session, SessionFactory, and Transaction classes. The native API gives you access to powerful capabilities such as custom caching strategies, HQL support, and other optimizations that go beyond the JPA specification. Use this approach when you need Hibernate's full feature set or are maintaining a legacy application built with Hibernate native APIs.

DAY 2:

In this session, we will dive into how to establish a database connection using Hibernate through JPA (Java Persistence API) implementation.

JPA and Server Clarification

- JPA is not a part of the JRE system library. The JRE (Java Runtime Environment) includes only core Java (Java SE) packages like java.util, java.io, java.lang, etc.
- JPA is part of the Java EE (Jakarta EE) specification, which defines enterprise-level features such as persistence, transactions etc…
- However, if you're using Apache Tomcat, you still won't get JPA support out of the box, because Tomcat is not a full Java EE server — it only supports the Servlet and JSP APIs.

- Therefore, to use JPA in a Tomcat-based application or in a simple console based application, you need to explicitly add the required dependencies yourself, including:
    1. The JPA API (javax.persistence-api.jar)
    2. A JPA implementation like Hibernate or EclipseLink

Note: Does Hibernate(JPA) Use JDBC Internally?

Yes, Hibernate uses JDBC internally to communicate with the database.

- Even though developers interact with the database using JPA or Hibernate APIs, these are just abstractions.
- Under the hood, Hibernate generates SQL queries and uses the JDBC API to send those queries to the database.
- So, all JPA implementations (including Hibernate) still depend on JDBC drivers and connections.

In short:

JPA → Hibernate → JDBC → Database

What Java EE (Jakarta EE) Contains

Java EE (now called Jakarta EE) is a full enterprise-level specification that

includes many APIs for building large, scalable, distributed applications. Here's a list of main components of Java EE:

Feature / API Package / Purpose
• JPA javax.persistence → ORM (database mapping)
• Servlets javax.servlet → Web request handling
• JSP (JavaServer Pages) javax.servlet.jsp → Templating for web UI
• EJB (Enterprise JavaBeans) javax.ejb → Business logic components
• JTA (Java Transaction API) javax.transaction → Transaction management
• CDI (Context & Dependency Injection) javax.enterprise.context, javax.inject
• JMS (Java Messaging Service) javax.jms → Messaging between components
• WebSocket API javax.websocket → Real time web communication
• Bean Validation javax.validation → Input validation (like @NotNull)
• JAX-RS (REST API) javax.ws.rs → Build RESTful web services
• JAX-WS (SOAP API) javax.xml.ws → Build SOAP web services
• Security APIs javax.security → Authentication, authorization

A Java EE server (like TomEE, Payara, WildFly) provides both:

- These APIs (interfaces/specs)
- Their working implementation at runtime

Before we try to get a connection using JPA and Hibernate, it's important to understand that: JPA is not a part of the JRE (Java Runtime Environment).

So, we must manually add the required libraries to our project.

Required JARs:
Hibernate Jars(including JPA):
https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.6.5.Final/hibernate-release-5.6.5.Final.zip/download

=====Program: Hibernate JPA Connection======

Launch.java

```
package com.mainapp;
import

javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Launch {
```

```
      public static void main(String[] args) {

 // Create EntityManagerFactory using the persistence unit name
 EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myJpaUnit");

 // Get EntityManager to interact with the database
 EntityManager em = emf.createEntityManager();

 System.out.println(em);
 System.out.println("JPA Connection Established Successfully!");

 // Close resources
 em.close();
 emf.close();
      }
}
```

persistence.xml


```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
 version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

 <persistence-unit name="myJpaUnit" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

 <properties>
 <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306" />
    <property    name="javax.persistence.jdbc.user"    value="root"    />
<property name="javax.persistence.jdbc.password" value="" />  <property
name="javax.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver" />
 </properties>
 </persistence-unit>
</persistence>
```

Program Explanation:

How Hibernate (JPA Connection Works in Implementation)

- **In JDBC, we manually create a**

connection using DriverManager.getConnection(...). But in Hibernate (via JPA), we don't do this directly. Instead, we let Hibernate handle it for us.

How do we know if we're connected to the Database in JPA?

- When `EntityManager em = emf.createEntityManager();` runs successfully, it means Hibernate has successfully connected to the database.

In this program:

- One EntityManagerFactory and one EntityManager object are created.
- EntityManagerFactory is responsible for initializing Hibernate and setting up the internal configuration – including connection management.
- Hibernate may open one JDBC connection behind the scenes, especially when you perform some database operation (like persist, find, query, etc.).

- Hibernate includes a basic built-in connection pool (if no external pool like HikariCP is configured).
- By default, Hibernate uses a simple connection provider (DriverManagerConnectionProviderImpl) which:
  1. Creates one internal pool manager object.
  2. Can open up to 20 JDBC connections by default (unless configured with hibernate.connection.pool_size).
  3. Does not reuse or manage connections efficiently like advanced pools (e.g., no timeout, validation, etc.).
- But this default pool is not recommended for production or high-traffic applications.

Why is META-INF Required?

- JPA (as per the official spec) looks for the persistence.xml file in the class path under META INF/. If the file is not placed there, JPA won't be

able to locate it. and you'll get runtime errors.

- This top part is like telling the XML parser: 'Hey, I'm writing a JPA configuration using version 2.2, and here's the official ruleset to validate it against.' Without it, Hibernate or your IDE might not understand or validate the structure properly.

```
<persistence
 version="2.2"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema
instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persi
stence

http://xmlns.jcp.org/xml/ns/persistence/persistence_2
_2.xsd">
```

==========

Think of EntityManagerFactory as the "boss" or "manager".

When your application starts, this boss sets up everything needed to interact with the database:

- **Loads configuration from persistence.xml**
- **Initializes Hibernate**
- **Sets up connection pooling**
- **Prepares mapping info (like entity classes)**

Once everything is ready, this boss gives you a worker object — the EntityManager — which actually does the job:

- **Sending queries**
- **Saving/updating data**
- **Starting/ending transactions**

You use EntityManager to talk to the database, but all the heavy setup work

was already done by EntityManagerFactory.

DAY 3: In this session of the Hibernate JPA tutorial, we dive into how to interact with the database using JPA's EntityManager interface. You'll learn how Hibernate communicates with the database behind the scenes using JPA, and how to perform essential CRUD operations — Create, Read, Update, and Delete — using EntityManager.

In Hibernate JPA implementation, we begin by understanding how to perform operations on a single row in the database using the EntityManager interface. These include the core CRUD operations: inserting a new row, reading an existing row, updating a record, and deleting a record — all through simple and intuitive methods like persist(), find(), merge(), and remove(). These methods allow us to interact with the database in an object oriented way, focusing on entities rather than SQL queries. Before diving into batch processing or complex queries, mastering these single-row operations helps build a strong foundation for using JPA effectively.

To interact with a database using Hibernate and JPA, we must first configure the database connection details and entity mappings. This setup tells Hibernate how to connect to the database and how to map Java classes to

database tables. There are three main approaches to perform this configuration:

1. Pure XML Configuration (No Annotations)

- All the mapping and configuration are done in XML files.

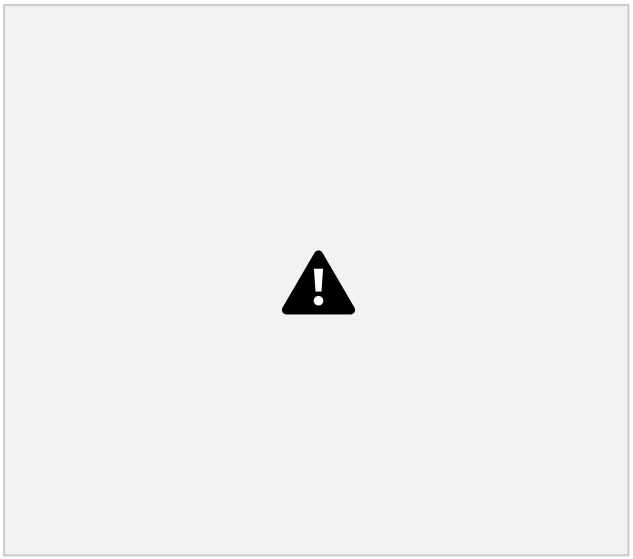
2. Hybrid (XML + Annotations)

- A combination of annotations in entity classes and some configuration in XML.

3. Annotation-Only Configuration (No XML)

- All configurations is done using annotations like @Entity, @Table, @Id, etc.

==========Program: PURE XML==========



```
package com.mainapp;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.entity.Employee;
public class Launch {

	public static void main(String[] args) {

		EntityManagerFactory ef = 
Persistence.createEntityManagerFactory("config");
		EntityManager em = ef.createEntityManager();

		//insert(em);
		//read(em);
```



```
		//update(em);
		//delete(em);


		em.close();
		ef.close();
	}

	private static void delete(EntityManager em) {

		Employee employee = em.find(Employee.class, 11);
		if(employee!=null) {

			EntityTransaction transaction = em.getTransaction();
```

```
                transaction.begin();

                em.remove(employee);

                transaction.commit();
                System.out.println("DATA DELETED SUCCESSFULLY");

            }else {
                System.out.println("DATA NOT FOUND");
            }

        }

    private static void update(EntityManager em) {

            Employee employee = em.find(Employee.class, 11);
            if(employee!=null) {

                EntityTransaction transaction = em.getTransaction();
                transaction.begin();

                employee.setEsalary(8000);
                em.merge(employee);

                transaction.commit();
                System.out.println("DATA UPDATED SUCCESSFULLY");

            }else {
                System.out.println("DATA NOT FOUND");
            }

        }

        private static void read(EntityManager em) {

            Employee employee = em.find(Employee.class, 11);
            System.out.println(employee);

        }
```



```
        private static void insert(EntityManager em) {

            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            Employee employee = new Employee(12, "kaju", "kkr", 2000);
            em.persist(employee);

            transaction.commit();
            System.out.println("DATA INSERTED SUCCESSFULLY");
```

```
	}
}


package com.entity;
//ENTITY
public class Employee {

	private int eid;
	private String ename;
	private String eaddress;
	private int esalary;

	public Employee() {
				//HIBERNATE WILL
	USE THIS
	CONSTRUCTOR
	}

	 public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public String getEaddress() {
		return eaddress;
	}
	public void setEaddress(String eaddress) {


		this.eaddress = eaddress;
	}
	public int getEsalary() {
		return esalary;
	}
```

```
	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

orm.xml

```
package com.entity;
//ENTITY
public class Employee {

	private int eid;
	private String ename;
	private String eaddress;

	private int esalary;

	public Employee() {
		//HIBERNATE WILL USE THIS CONSTRUCTOR
	}

	 public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public String getEaddress() {
```

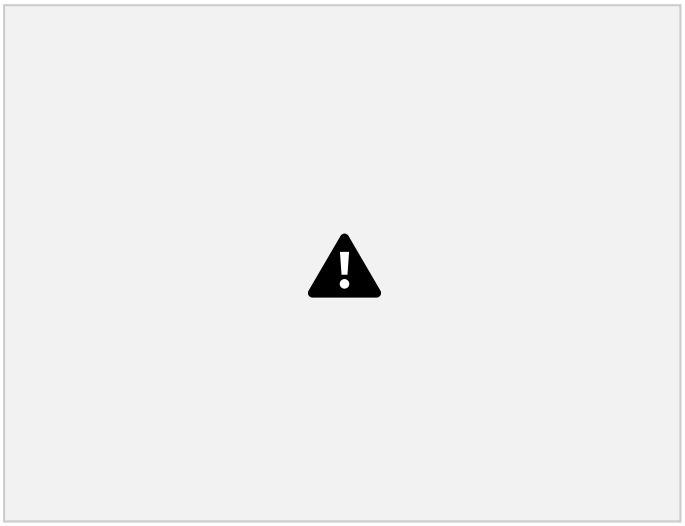
```
		return eaddress;
	}
	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}
	public int getEsalary() {
		return esalary;
	}
	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
 version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"


 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">

 <persistence-unit name="config">
 <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<mapping-file>META-INF/orm.xml</mapping-file>

 <properties>
 <property name="javax.persistence.jdbc.url"
 value="jdbc:mysql://localhost:3306/hiber"/>

 <property name="javax.persistence.jdbc.user"
 value="root"/>

 <property name="javax.persistence.jdbc.password"
 value=""/>

 <property name="javax.persistence.jdbc.driver"
 value="com.mysql.cj.jdbc.Driver"/>

 <property name="hibernate.hbm2ddl.auto"
 value="update"/> <!-- u can use create also (DELETE  OLD & CREATE NEW
TABLE EVERYTIME) -->

 <property name="hibernate.dialect"
 value="org.hibernate.dialect.MySQL8Dialect"/>
```

```
<property name="hibernate.show_sql"
value="true"/>

</properties>

</persistence-unit>
</persistence>
```

==========Program: HYBRID (XML + ANNO)==========



```
package com.mainapp;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.entity.Employee;
public class Launch {

	public static void main(String[] args) {

		EntityManagerFactory ef =
Persistence.createEntityManagerFactory("config");
		EntityManager em = ef.createEntityManager();

		insert(em);
		//read(em);
		//update(em);
		//delete(em);


		em.close();
		ef.close();
```

```
}

private static void delete(EntityManager em) {

```



```
	Employee employee = em.find(Employee.class, 11);
	if(employee!=null) {

		EntityTransaction transaction = em.getTransaction();
		transaction.begin();

		em.remove(employee);

		transaction.commit();
		System.out.println("DATA DELETED SUCCESSFULLY");

	}else {
		System.out.println("DATA NOT FOUND");
	}

}

private static void update(EntityManager em) {

	Employee employee = em.find(Employee.class, 11);
	if(employee!=null) {

		EntityTransaction transaction = em.getTransaction();
		transaction.begin();
```



```
		employee.setEsalary(8000);
		em.merge(employee);

		transaction.commit();
		System.out.println("DATA UPDATED SUCCESSFULLY");

	}else {
		System.out.println("DATA NOT FOUND");
	}

}

private static void read(EntityManager em) {

	Employee employee = em.find(Employee.class, 11);
	System.out.println(employee);

}

private static void insert(EntityManager em) {
```

```
            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            Employee employee = new Employee(12, "kaju", "kkr", 2000);
            em.persist(employee);

            transaction.commit();
```



```
            System.out.println("DATA INSERTED SUCCESSFULLY");

      }
}
```

```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee") //OPTIONAL
public class Employee {

      @Id
      @Column(name = "id") //OPTIONAL
      private int eid;

      @Column(name = "name" , length = 30) //OPTIONAL
      private String ename;

      @Column(name = "address" , length = 200) //OPTIONAL
      private String eaddress;

      @Column(name =
      "salary")
      //OPTIONAL
      private int esalary;

      public Employee() {
            //HIBERNATE WILL USE THIS CONSTRUCTOR
      }

       public Employee(int eid, String ename, String eaddress, int esalary)
{
            super();
            this.eid = eid;
            this.ename = ename;
            this.eaddress = eaddress;
            this.esalary = esalary;
      }
```

```
	public int getEid() {
		return eid;
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
```



```
	public String getEaddress() {
		return eaddress;
	}
	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}
	public int getEsalary() {
		return esalary;
	}
	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
 version="2.2"
```

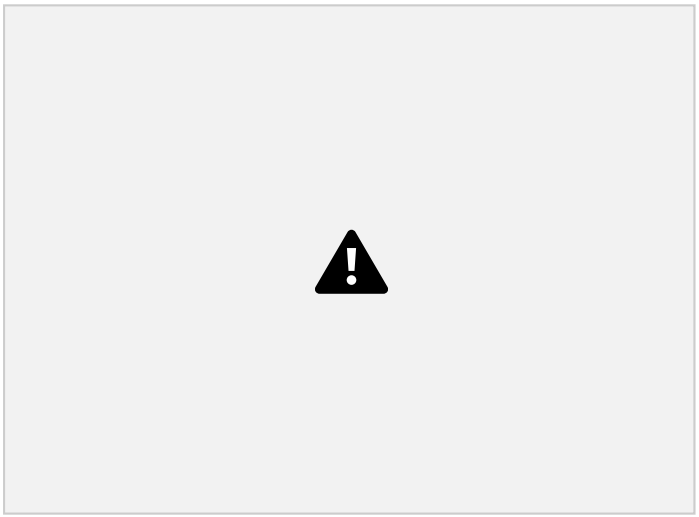


```
 xmlns="http://xmlns.jcp.org/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">

 <persistence-unit name="config">
 <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>com.entity.Employee</class>

 <properties>
```

```
 <property name="javax.persistence.jdbc.url"
 value="jdbc:mysql://localhost:3306/hiber"/>

 <property name="javax.persistence.jdbc.user"
 value="root"/>

 <property name="javax.persistence.jdbc.password"
 value=""/>

 <property name="javax.persistence.jdbc.driver"
 value="com.mysql.cj.jdbc.Driver"/>

 <property name="hibernate.hbm2ddl.auto"
 value="update"/> <!-- u can use create also (DELETE  OLD & CREATE NEW
TABLE EVERYTIME) -->

 <property name="hibernate.dialect"
 value="org.hibernate.dialect.MySQL8Dialect"/>
```


```
 <property name="hibernate.show_sql"
 value="true"/>

 </properties>

 </persistence-unit>
 </persistence>
```

==========Program: ANNOTATION (NO XML)==========



```
package com.mainapp;
import
java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import javax.persistence.EntityManager;
```

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import org.hibernate.jpa.HibernatePersistenceProvider;
import com.entity.Employee;
import com.entity.PersistenceUnitInfoImpl;
public class Launch {

	public static void main(String[] args) {

		Properties properties=new Properties();
		InputStream is = null;
		try {

	is=Launch.class.getClassLoader().getResourceAsStream("config.propert
ies");
			if(is==null) {
				System.out.println("FILE NOT FOUND");
				return;
			}
```



```
			properties.load(is);


		} catch (IOException e) {
			e.printStackTrace();
		}
		finally {
			if(is!=null) {
				try {
					is.close();
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		}

		Map<String, String> map=new HashMap<String, String>();
		 for( String key : properties.stringPropertyNames() ) {
			map.put(key, properties.getProperty(key));
		}
		System.out.println(map);

		 EntityManagerFactory ef = new HibernatePersistenceProvider()
				 .createContainerEntityManagerFactory(new
PersistenceUnitInfoImpl(), map);
```

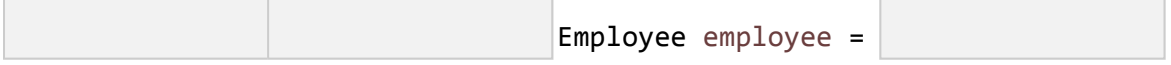


```
EntityManager em =
				ef.createEntityManager();

		insert(em);
```

```
		//read(em);
		//update(em);
		//delete(em);

		em.close();
		ef.close();
	}

	private static void delete(EntityManager em) {

		Employee employee = em.find(Employee.class, 11);
		if(employee!=null) {

			EntityTransaction transaction = em.getTransaction();
			transaction.begin();

			em.remove(employee);

			transaction.commit();
			System.out.println("DATA DELETED SUCCESSFULLY");

		}else {
			System.out.println("DATA NOT FOUND");
```



```
		}

	}

	private static void update(EntityManager em) {

		Employee employee = em.find(Employee.class, 11);
		if(employee!=null) {

			EntityTransaction transaction = em.getTransaction();
			transaction.begin();

			employee.setEsalary(8000);
			em.merge(employee);

			transaction.commit();
			System.out.println("DATA UPDATED SUCCESSFULLY");

		}else {
			System.out.println("DATA NOT FOUND");
		}

	}

	private static void read(EntityManager em) {
```



```
Employee employee =
		em.find(Employee.class, 11);
```

```
            System.out.println(employee);

      }

      private static void insert(EntityManager em) {

            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            Employee employee = new Employee(450, "kaju", "kkr", 2000);
            em.persist(employee);

            transaction.commit();
            System.out.println("DATA INSERTED SUCCESSFULLY");

      }
}
```

```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```



```
@Table(name = "xemployee") //OPTIONAL
public class Employee {

      @Id
      @Column(name = "id") //OPTIONAL
      private int eid;

      @Column(name = "name" , length = 30) //OPTIONAL
      private String ename;

      @Column(name = "address" , length = 200) //OPTIONAL
      private String eaddress;

      @Column(name = "salary") //OPTIONAL
      private int esalary;

      public Employee() {
            //HIBERNATE WILL USE THIS CONSTRUCTOR
      }

      public Employee(int eid, String ename, String eaddress, int
esalary) {
            super();
```

```
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public String getEaddress() {
		return eaddress;
	}
	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}
	public int getEsalary() {
		return esalary;
```



```
	}
	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}


package com.entity;
import java.net.URL;
import java.util.Arrays;
import java.util.List;
import java.util.Properties;
import javax.persistence.SharedCacheMode;
import javax.persistence.ValidationMode;
import javax.persistence.spi.ClassTransformer;
```

```
import javax.persistence.spi.PersistenceUnitInfo;
import javax.persistence.spi.PersistenceUnitTransactionType;
import javax.sql.DataSource;

public class PersistenceUnitInfoImpl implements
        PersistenceUnitInfo {
@Override
public void addTransformer(ClassTransformer arg0) {
      // TODO Auto-generated method stub

}

@Override
public boolean excludeUnlistedClasses() {
      // TODO Auto-generated method stub
      return false;
}

@Override
public ClassLoader getClassLoader() {
      // TODO Auto-generated method stub
      return null;
}

@Override
public List<URL> getJarFileUrls() {
      // TODO Auto-generated method stub
      return null;
}

@Override
```



```
public DataSource getJtaDataSource() {
      // TODO Auto-generated method stub
      return null;
}

@Override
public List<String> getManagedClassNames() {
      // TODO Auto-generated method stub
      return Arrays.asList("com.entity.Employee");
}

@Override
public List<String> getMappingFileNames() {
      // TODO Auto-generated method stub
      return null;
}

@Override
public ClassLoader getNewTempClassLoader() {
```

```
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public DataSource getNonJtaDataSource() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public String getPersistenceProviderClassName() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public String getPersistenceUnitName() {
		// TODO Auto-generated method stub
		return "config";
	}

	@Override
	public URL getPersistenceUnitRootUrl() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public String getPersistenceXMLSchemaVersion() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
```

```
	public Properties getProperties() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public SharedCacheMode getSharedCacheMode() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public PersistenceUnitTransactionType getTransactionType() {
		// TODO Auto-generated method stub
```

```
		return null;
	}

	@Override
	public ValidationMode getValidationMode() {
		// TODO Auto-generated method stub
		return null;
	}

}
```

config.properties

```
javax.persistence.jdbc.url=jdbc:mysql://localhost:3306/hiber
javax.persistence.jdbc.user=root
javax.persistence.jdbc.password=
javax.persistence.jdbc.driver=com.mysql.cj.jdbc.Driver
hibernate.hbm2ddl.auto=update
hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
hibernate.show_sql=true
```

DAY 4:

Bulk Operations in Hibernate JPA Implementation

Till now, we have worked with single row operations using three configuration approaches:

- **XML-based**
- **Hybrid (XML + Annotations)**
- **Annotation-based**

We explored JPA methods like:

- **persist() – for creating new records**
- **find() – for retrieving records**
- **merge() – for updating**
- **remove() – for deleting**

Now, we move to bulk operations, i.e., performing Create, Read, Update, Delete (CRUD) on multiple records at once. Hibernate JPA allows bulk

operations through three main approaches:

Native SQL

- Write raw SQL using actual table and column names.
- Directly executed by the database engine.
- Supports all CRUD operations: INSERT, SELECT, UPDATE, DELETE

Best suited when:

- You need DB-specific features (e.g., MySQL keywords, stored procedures).
- You want full control over SQL.

JPQL (Java Persistence Query Language)

- Object-oriented query language using entity class and field names.
- Translated to SQL by the JPA provider (like Hibernate).
- Does not support INSERT
- Supports SELECT, UPDATE, DELETE
- Best suited for writing portable and clean logic across databases.

Criteria API

- Pure Java way to build queries using CriteriaBuilder.
- Supports: SELECT, UPDATE, DELETE
- Does not support INSERT

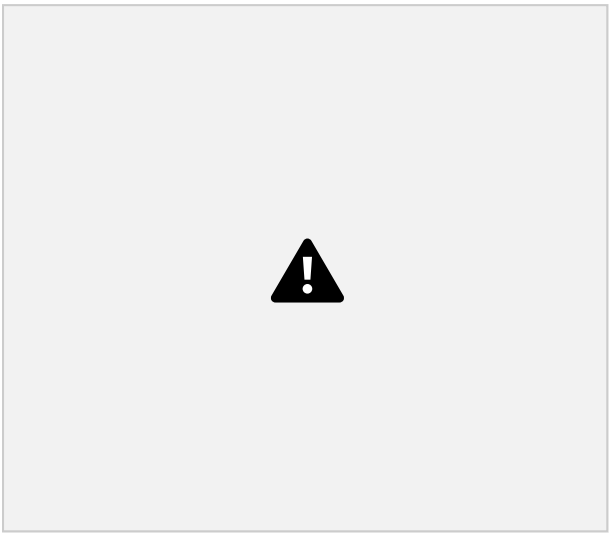
Best suited for:

- Dynamic queries
- Type safety
- Query-free code

Named Queries in JPA – Quick Notes

- **Predefined queries using annotations – reused multiple times.**
- **Defined at Entity level using @NamedQuery or @NamedNativeQuery.**
- **Improves code readability, performance (can be compiled at startup).**

========Program:CRUD USING SQL JPQL & CRITERIA==========



```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedNativeQuery;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee") //OPTIONAL
@NamedQuery(name = "deleteJPQL" , query = "delete from Employee where
eid<=:eid")
@NamedNativeQuery(name = "deleteSQL" , query = "delete from xemployee
where id<=:id")
public class Employee {
```

```
    @Id
    @Column(name = "id") //OPTIONAL
    private int eid;
```

```
	@Column(name = "name" , length = 30) //OPTIONAL
	private String ename;

	@Column(name = "address" , length = 200) //OPTIONAL
	private String eaddress;

	@Column(name = "salary") //OPTIONAL
	private int esalary;

	public Employee() {
		//HIBERNATE WILL USE THIS CONSTRUCTOR
	}

	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
	}

	 public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress =
		eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	public void setEname(String ename) {
		this.ename = ename;
	}
	public String getEaddress() {
		return eaddress;
	}
	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}
	public int getEsalary() {
		return esalary;
	}
	public void setEsalary(int esalary) {
```

```
            this.esalary = esalary;
      }

      @Override
      public String toString() {
            return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
      }
}


package com.mainapp;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
import com.entity.Employee;

public class Launch {

      // NATIVE SQL
      public static void main(String[] args) {

            EntityManagerFactory ef =

Persistence.createEntityManagerFactory("config");
            EntityManager em = ef.createEntityManager();

            //insert(em);
             //read(em);
             //update(em);
             delete(em);

            em.close();
            ef.close();
      }

      private static void delete(EntityManager em) {
            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            Query query = em.createNamedQuery("deleteSQL");
            query.setParameter("id", 117);

            query.executeUpdate();

            transaction.commit();
            System.out.println("BULK DATA DELETED");
 em.close();
      }

      private static void update(EntityManager em) {
            EntityTransaction transaction = em.getTransaction();
```

```
		transaction.begin();

		String sql = "update xemployee set salary=? where id>=?";
		Query query = em.createNativeQuery(sql);
		query.setParameter(1, 98989898);
		query.setParameter(2, 12347);

		query.executeUpdate();

		transaction.commit();
		System.out.println("BULK DATA UPDATED");
		em.close();
	}

	private static void read(EntityManager em) {

// String sql = "select * from xemployee";
// Query query = em.createNativeQuery(sql);
// List<Object[]> list = query.getResultList();
//
// for(Object[] orr : list) {
// for(Object o : orr ) {
// System.out.print(o+" ");
// }
// System.out.println();
```

```
		String sql =
		"select * from xemployee";
		Query query = em.createNativeQuery(sql,Employee.class);
		List<Employee> list = query.getResultList();

		for(Employee e : list) {
			System.out.println(e);
		}

		em.close();

	}

	private static void insert(EntityManager em) {

		EntityTransaction transaction = em.getTransaction();
		transaction.begin();

		 String sql = "insert into xemployee(id,name,address,salary)
values(?,?,?,?)";

		for (int i = 1; i <= 10; i++) {

			Query nativeQuery = em.createNativeQuery(sql);
```

```
                nativeQuery.setParameter(1, 12345+i);
                nativeQuery.setParameter(2, "xyzijkl");
                nativeQuery.setParameter(3, "adddrrr");
                nativeQuery.setParameter(4, 676754+i);
```



```
                nativeQuery.executeUpdate();
            }
            transaction.commit();
            em.close();
            System.out.println("BULK DATA INSERTED");
      }
}
```

```
package com.mainapp;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
import com.entity.Employee;
public class Launch2 {

      // JPQL
      public static void main(String[] args) {

            EntityManagerFactory ef =
```



```
Persistence.createEntityManagerFactory("config");
            EntityManager em = ef.createEntityManager();

             //insert(em);
             //read(em);
             //update(em);
            delete(em);

            em.close();
            ef.close();
      }

      private static void delete(EntityManager em) {
// EntityTransaction transaction = em.getTransaction();
// transaction.begin();
//
// String sql = "delete from Employee where eid>=:eid";
// Query query = em.createQuery(sql);
// query.setParameter("eid", 900);
```

```
//
// query.executeUpdate();
//
// transaction.commit();
// System.out.println("BULK DATA DELETED");
// em.close();


            EntityTransaction transaction = em.getTransaction();
```



```
            transaction.begin();

            Query query = em.createNamedQuery("deleteJPQL");
            query.setParameter("eid", 13);

            query.executeUpdate();

            transaction.commit();
            System.out.println("BULK DATA DELETED");
            em.close();
      }

      private static void update(EntityManager em) {
            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

             String jpql = "update Employee set esalary=:esalary where
eid>=:eid";
            Query query = em.createQuery(jpql);
            query.setParameter("esalary", 98765);
            query.setParameter("eid", 900);

            query.executeUpdate();

            transaction.commit();
            System.out.println("BULK DATA UPDATED");
```



```
            em.close();
      }

      private static void read(EntityManager em) {


              String jpql = "select e from Employee e"; //COMPULSORY
ALIASING
            Query query = em.createQuery(jpql,Employee.class);
            List<Employee> list = query.getResultList();

            for (Employee e : list) {
                  System.out.println(e);
            }
```

```
			em.close();

		}

		private static void insert(EntityManager em) {

			EntityTransaction transaction = em.getTransaction();
			transaction.begin();

			for (int i = 1; i <= 10; i++) {
				Employee employee = new Employee(900+i, "jpqlname",
"jpqladdr", 1000);
				em.persist(employee);
```



```
				if(i%5==0) {
					em.flush(); //CACHE MEMORY---------------------
>DATABASE HIT(WAIT FOR CHANGE)
				}
			}

			transaction.commit();
			em.close();
			System.out.println("BULK DATA INSERTED");
		}
}


package com.mainapp;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaDelete;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.CriteriaUpdate;
import
```

```
javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import com.entity.Employee;

public class Launch3 {

		// JPQL
		public static void main(String[] args) {
```

```
            EntityManagerFactory ef =
Persistence.createEntityManagerFactory("config");
            EntityManager em = ef.createEntityManager();

            // insert(em);
            //read(em);
             //update(em);
             delete(em);

            em.close();
            ef.close();
      }

      private static void delete(EntityManager em) {

            EntityTransaction transaction = em.getTransaction();
```



```
            transaction.begin();

            CriteriaBuilder criteriaBuilder =
em.getCriteriaBuilder();
            CriteriaDelete<Employee> cd =
criteriaBuilder.createCriteriaDelete(Employee.class);
            Root<Employee> root = cd.from(Employee.class);

            cd.where(criteriaBuilder.lessThan(root.get("eid"), 905));

            Query query = em.createQuery(cd);
            query.executeUpdate();

            transaction.commit();
            System.out.println("BULK DATA DELETED");
            em.close();
      }

      private static void update(EntityManager em) {
            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            CriteriaBuilder criteriaBuilder =
em.getCriteriaBuilder();
```



```
            CriteriaUpdate<Employee> cu =
criteriaBuilder.createCriteriaUpdate(Employee.class);
            Root<Employee> root = cu.from(Employee.class);
```

```
            cu.set("esalary", 6000); //set esalary=6000
             cu.where(criteriaBuilder.lessThan(root.get("eid"), 905));
//eid<905

            Query query = em.createQuery(cu);
            query.executeUpdate();

            transaction.commit();
            System.out.println("BULK DATA UPDATED");
            em.close();
      }

      private static void read(EntityManager em) {

            // select * from employee

// CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder(); //
// CriteriaQuery<Employee> cq =
criteriaBuilder.createQuery(Employee.class);//READ CRITERIA
```



```
// Root<Employee> from = cq.from(Employee.class); //from
Employee
// cq.select(from); //select * from Employee
//
// TypedQuery<Employee> query = em.createQuery(cq);
// List<Employee> list = query.getResultList();
//
// System.out.println(list);
//
// em.close();

            //select id name where name like 'j%' and id>905

            CriteriaBuilder criteriaBuilder =
em.getCriteriaBuilder();

            CriteriaQuery<Employee> cq =
criteriaBuilder.createQuery(Employee.class);
             Root<Employee> root = cq.from(Employee.class);

             Predicate p1 = criteriaBuilder.like(root.get("ename"),
"j%"); //name like 'j%'
            Predicate p2 =
criteriaBuilder.greaterThan(root.get("eid"), 905); // id>905
```

```
   cq.multiselect(root.get("eid"),root.get("ename")).where(criter
iaBuilder.and(p1,p2));

            List<Employee> list = em.createQuery(cq).getResultList();
            System.out.println(list);

            em.close();

      }

      private static void insert(EntityManager em) {

            EntityTransaction transaction = em.getTransaction();
            transaction.begin();

            for (int i = 1; i <= 10; i++) {
                  Employee employee = new Employee(900 + i,
"jpqlname", "jpqladdr", 1000);
                  em.persist(employee);

                  if (i % 5 == 0) {
                        em.flush(); // CACHE MEMORY----------------
--->DATABASE HIT(WAIT FOR CHANGE)
```



```
                  }
            }

            transaction.commit();
            em.close();
            System.out.println("BULK DATA INSERTED");
      }
}
```

DAY 5:

Bulk Operations in Hibernate JPA Implementation

Till now, we have learned how to implement Hibernate using JPA and performed CRUD operations using different approaches:

- **Single Entity CRUD operations**
- **JPQL (Java Persistence Query Language)**
- **Native SQL Queries**
- **Criteria API (JPA Criteria)**

Next Step:

- Now, we are starting with the Hibernate Native API to understand how to work directly with Hibernate's core interfaces such as Session, Transaction, and Query, without relying on the JPA abstraction.
- The Hibernate Native API is powerful because it provides direct access to Hibernate's core functionalities, offering more control and flexibility than JPA.

In Hibernate Native API, the first step is to learn how to establish a connection with the database. This can be done in two ways:

- Using XML configuration (hibernate.cfg.xml)
- Without XML, by using Java-based configuration (i.e., setting properties programmatically in code).

TASK: CONFIG.properties

Connection Code Explained:

1. Configuration

- It represents the Hibernate configuration settings.
- Used to load database connection info and Hibernate mappings.

2. SessionFactory

- It is a heavyweight object.
- Created once per application and reused.

Responsible for:

1. Holding Hibernate config
2. Creating Session objects
3. Managing connection pooling and caching

3.Session

- A lightweight, short-lived object used to interact with the database.

- Represents a single unit of work.
- Used to Perform CRUD operations

In Hibernate Native API, we can interact with the database in two main ways:

1.Single Row Operation

- Interacts with one entity/object at a time

2.Bulk Row Operation

- Interacts with multiple records at once

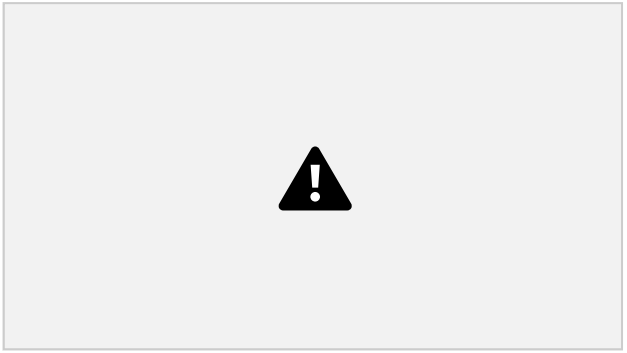
Single Row Operation

- In Hibernate Native API, Single Row Operations refer to interacting with one record (object) at a time in the database. These are the most common and straightforward operations used for basic CRUD tasks.

- Hibernate provides methods like get(), load() to fetch a single record, save() to insert, update() to modify, and delete() to remove an entity.

- When a record is fetched using get(), it becomes part of the persistence context, and any changes made to it are automatically tracked by Hibernate. Upon committing the transaction, Hibernate updates the database if any changes are detected. This approach is simple, clean, and ideal for use cases where you're dealing with one entity at a time.

```
======Program: Connection Using Hibernate Native
API(XML)============
```



```
package com.mainapp;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class Launch {

	public static void main(String[] args) {


		//Load configuration from hibernate.cfg.xml
		Configuration configuration = new Configuration();
		configuration.configure();

		 //SessionFactory : SETUP READY (heavy weight object (pool,
mappings, Session ))
		SessionFactory sf = configuration.buildSessionFactory();

		//To interact with DB we use Session
		Session session = sf.openSession();

		System.out.println(session);

	}
}
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!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.cj.jdbc.Driver</property>
		<property
name="connection.url">jdbc:mysql://localhost:3306</property>
```

```
            <property name="connection.username">root</property>
            <property name="connection.password"></property>
      </session-factory>
</hibernate-configuration>
```

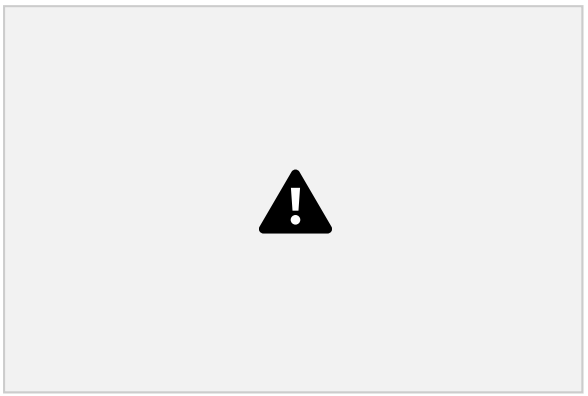
=========Program: Connection Using Hibernate Native API(ANNOTATION) ==========

```
package com.mainapp;
import java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
public class Launch {

      public static void main(String[] args) {

            Properties properties = new Properties();
            properties.put(Environment.URL,
"jdbc:mysql://localhost:3306");
            properties.put(Environment.USER, "root");
            properties.put(Environment.PASS, "");
            properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
```



```

            Configuration configuration = new Configuration();
            configuration.setProperties(properties);

            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();

            System.out.println(session);
      }
}
```

=======Program: CRUD Using Hibernate Native API(XML)==========



```
package com.mainapp;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.entity.Employee;

public class Launch {

	public static void main(String[] args) {

		Configuration configuration = new Configuration();
		configuration.configure();

		SessionFactory sf = configuration.buildSessionFactory();

		Session session = sf.openSession();

		// insert(session);
		//read(session);
		//update(session);
		delete(session);

		session.close();
		sf.close();
```

```
}

private static void delete(Session session) {

	Transaction transaction = session.getTransaction();

	Employee employee = session.get(Employee.class, 11);

	if(employee!=null) {

		transaction.begin();
		session.delete(employee);
		System.out.println("DATA DELETED");
		transaction.commit();

	}
	else {
		System.out.println("DATA NOT FOUND");
	}
}

private static void update(Session session) {

	Transaction transaction = session.getTransaction();
	transaction.begin();
	Employee employee = session.get(Employee.class, 11);

	if(employee!=null) {

		employee.setEsalary(9000);
		session.update(employee);
		System.out.println("DATA UPDATED");

	}else {
		System.out.println("DATA NOT FOUND");
	}

	transaction.commit();
}

private static void read(Session session) {

	Employee employee = session.get(Employee.class, 11);
	System.out.println(employee);
}

private static void insert(Session session) {

	System.out.println(session);

	Employee employee = new Employee(11, "name", "address", 1000);
	Transaction transaction = session.getTransaction();

	transaction.begin();
```

```
			session.save(employee);

			transaction.commit();
			System.out.println("DATA INSERTED");
		}
	}
	package com.entity;
	public class Employee {

		private int eid;
		private String ename;
		private String eaddress;
		private int esalary;

		public Employee() {
		}

		public Employee(int eid, String ename) {
			super();
			this.eid = eid;
			this.ename = ename;
		}

		 public Employee(int eid, String ename, String eaddress, int esalary)
	{
```



```
			super();
			this.eid = eid;
			this.ename = ename;
			this.eaddress = eaddress;
			this.esalary = esalary;
		}

		public int getEid() {
			return eid;
		}

		public void setEid(int eid) {
			this.eid = eid;
		}

		public String getEname() {
			return ename;
		}

		public void setEname(String ename) {
			this.ename = ename;
		}
```

```
	public String getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
```



```
		this.eaddress = eaddress;
	}

	public int getEsalary() {
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

}
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!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.cj.jdbc.Driver</property>
		<property
name="connection.url">jdbc:mysql://localhost:3306/hiber</property>
		<property name="connection.username">root</property>
		<property name="connection.password"></property>
		<property name="hibernate.hbm2ddl.auto">update</property>
		<property name="hibernate.show_sql">true</property>

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

	</session-factory>
</hibernate-configuration>
```

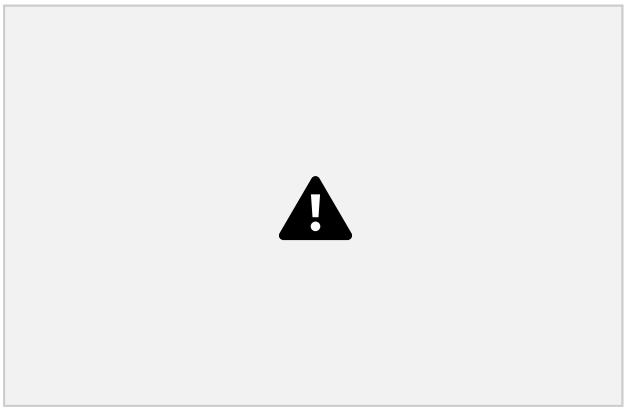
Employee.cfg.xml

```
<?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.entity.Employee" table="xemployee" >
 <id name="eid" column="id" ></id>
 <property name="ename" column="name" ></property>
```

```
 <property name="eaddress" column="address" ></property>
 <property name="esalary" column="salary" ></property>
 </class>

 </hibernate-mapping>
```

==========Program: CRUD Using Hibernate Native API(ANNOTATION)============



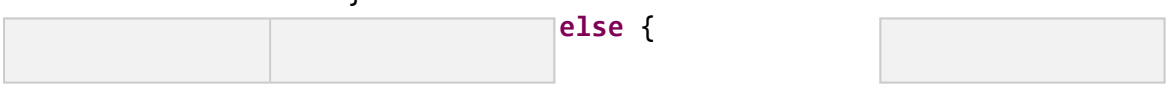
```
package com.mainapp;
import
```

```
java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import com.entity.Employee;
public class Launch {

      public static void main(String[] args) {

            Properties properties = new Properties();
            properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
            properties.put(Environment.USER, "root");
```

```
            properties.put(Environment.PASS, "");
            properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
            properties.put(Environment.HBM2DDL_AUTO, "update");
            properties.put(Environment.SHOW_SQL, "true");


            Configuration configuration = new Configuration();
            configuration.setProperties(properties);
            configuration.addAnnotatedClass(Employee.class);

            SessionFactory sessionFactory =
configuration.buildSessionFactory();
```



```
            Session session = sessionFactory.openSession();

            //insert(session);
             read(session);
            // update(session);
            // delete(session);

            session.close();
            sessionFactory.close();

      }

      private static void delete(Session session) {

            Transaction transaction = session.getTransaction();

            Employee employee = session.get(Employee.class, 11);

            if(employee!=null) {

                  transaction.begin();
                  session.delete(employee);
                  System.out.println("DATA DELETED");
                  transaction.commit();

            }
```



```
                  else {

                  System.out.println("DATA NOT FOUND");
            }
      }

      private static void update(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();
            Employee employee = session.get(Employee.class, 11);
```

```
            if(employee!=null) {

                  employee.setEsalary(9000);
                  session.update(employee);
                  System.out.println("DATA UPDATED");

            }else {
                  System.out.println("DATA NOT FOUND");
            }

            transaction.commit();
      }

      private static void read(Session session) {

            Employee employee = session.get(Employee.class, 11);
            System.out.println(employee);
```



```
      }

      private static void insert(Session session) {

            System.out.println(session);

            Employee employee = new Employee(11, "name", "address", 1000);
            Transaction transaction = session.getTransaction();

            transaction.begin();

            session.save(employee);

            transaction.commit();
            System.out.println("DATA INSERTED");
      }
}
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee")
public class
```

```
      @Id
      @Column(name = "id")
      private int eid;

      @Column(name = "name")
      private String ename;
```

```
	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public Employee() {
	}

	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
	}

	public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
```



```
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public String getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	public
	int getEsalary() {
```

```
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

DAY 6:

get() vs load()

Point get()

load()

1.Fetch Type Eager — hits DB immediately Lazy — returns proxy, hits DB when needed

2.Return Value Returns actual object or null if not found Returns proxy object; throws exception if not found

3.Exception Returns null if not found Throws ObjectNotFoundException if accessed but not in DB

4.Use Case When you're not sure if record exists When you're sure the record exists

5.Performance Slightly slower Faster (lazy load)

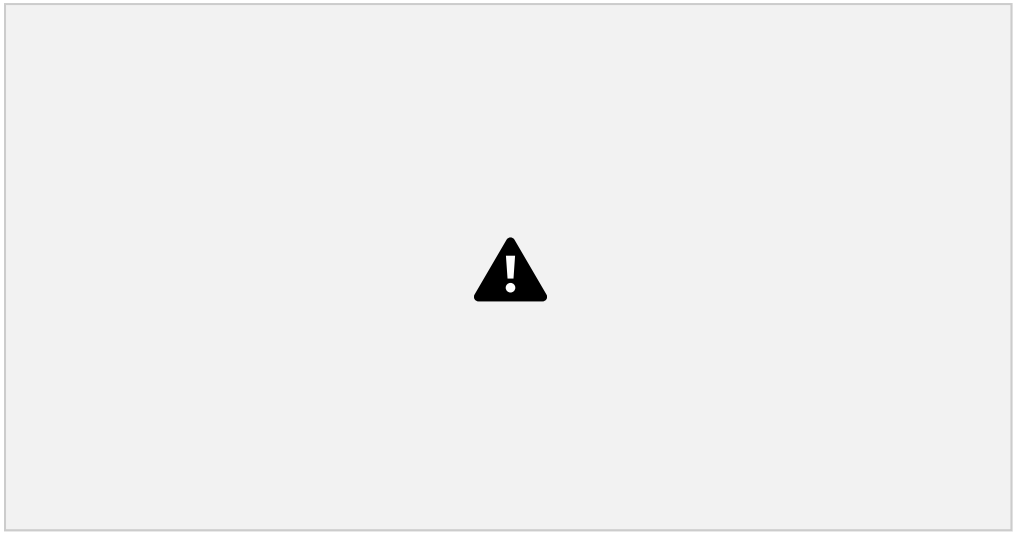
Why Regular Hibernate Session is Not Efficient for Bulk Operations

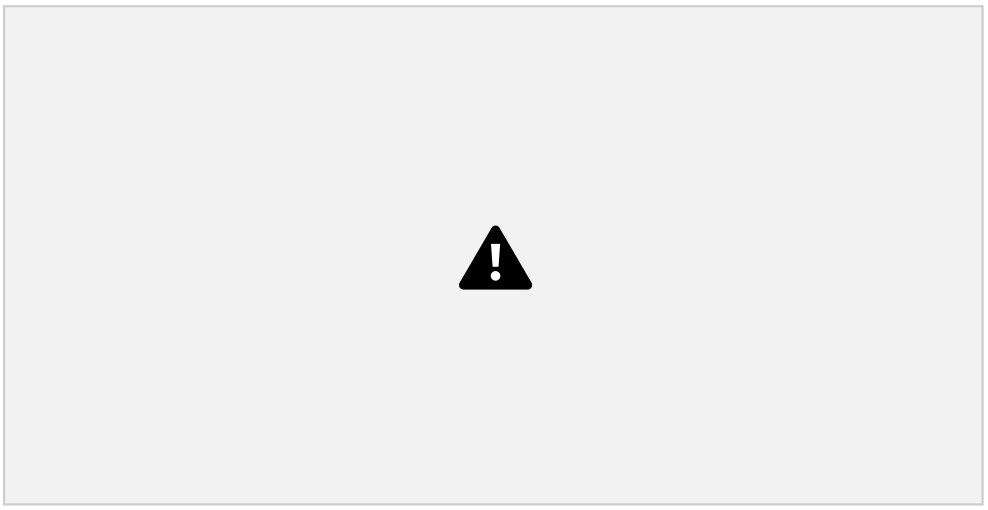
- **Hibernate's regular Session maintains a First-Level Cache (FLC) and performs automatic dirty checking to track object state changes. While this is helpful in normal CRUD operations, it becomes inefficient and memory-heavy during bulk operations like insert, update, delete, and read.**

What is First-Level Cache (FLC) in Hibernate?

- **First-Level Cache (FLC) is the default cache mechanism in Hibernate that stores objects within the scope of a session. Every Hibernate**

Session has its own FLC, and it's enabled by default — you don’t need to configure it.





When Can FLC Become a Problem?

- Especially during bulk operations, like inserting/updating thousands of rows:
- All objects are cached in memory.
- Hibernate tracks all objects for changes (dirty checking).
- Can lead to:
  - High memory usage
  - OutOfMemoryError
  - Slower performance

Managing First-Level Cache

- session.clear() – clears the entire cache.
- session.evict(Object) – removes a specific object from the cache.
- session.flush() – pushes changes from cache to the database.

Bulk Reads – Why Session is Inefficient

- Fetching thousands of records using get() or load() stores all entities in memory.
- Hibernate caches them in the First-Level Cache.
  - Problem: Heavy memory load and performance slowdown with large datasets.

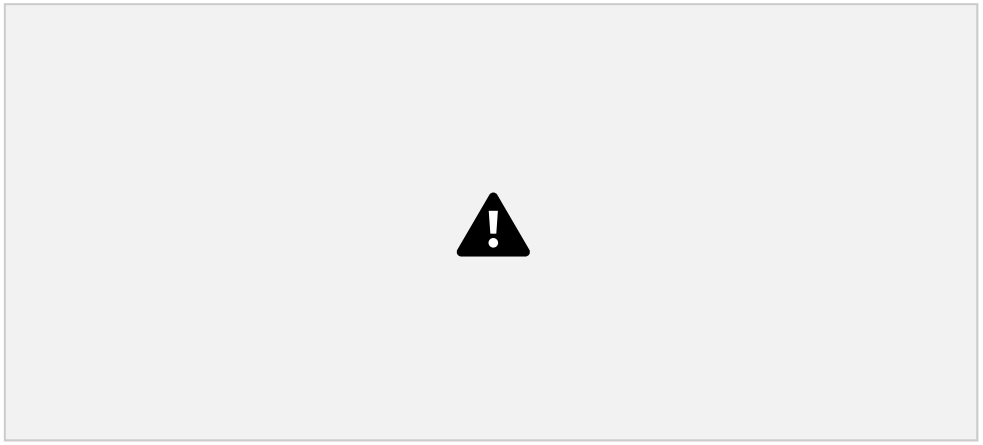
Bulk Inserts – Why Session is Inefficient

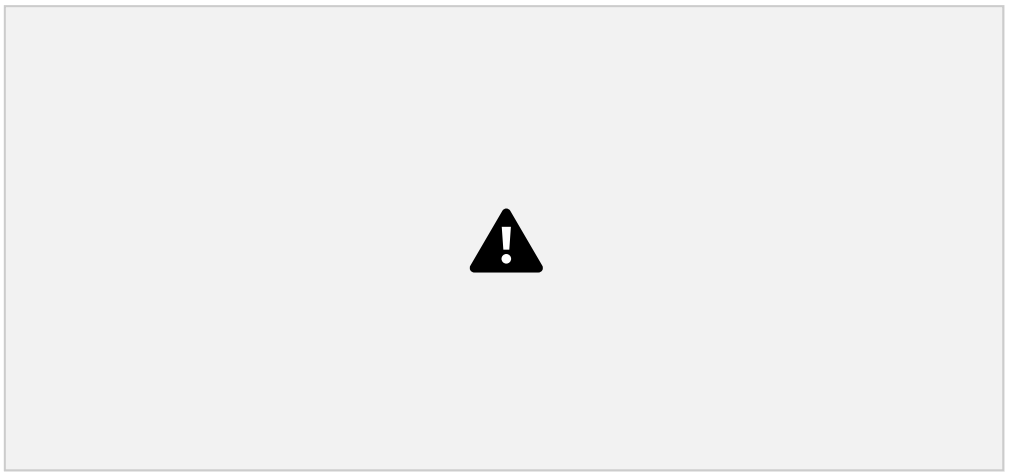
- When you use save() or persist() repeatedly:
- Hibernate stores every inserted object in the session's First-Level Cache.
- Over time, memory usage increases significantly.
- If not managed with flush() and clear(), this can lead to OutOfMemoryError.
  - Problem: Memory bloat due to retained object references after each insert.

Bulk Updates – Why Session is Inefficient

- Every loaded object is tracked in the session.
  - Hibernate performs dirty checking — it compares old vs new values before updating.

  - All objects remain in memory until you manually clear() the session.
- Problem: High memory usage and CPU overhead for unnecessary comparisons.





Bulk Deletes – Why Session is Inefficient

- When deleting via session.delete(), you typically load each entity using get() or load().
- These entities are cached.
- Though no dirty checking is performed for deletion logic itself, the loaded entities still consume memory.
- Problem: Extra memory usage from caching each object just to delete it.

Solutions for Efficient Bulk Operations in Hibernate: Use StatelessSession

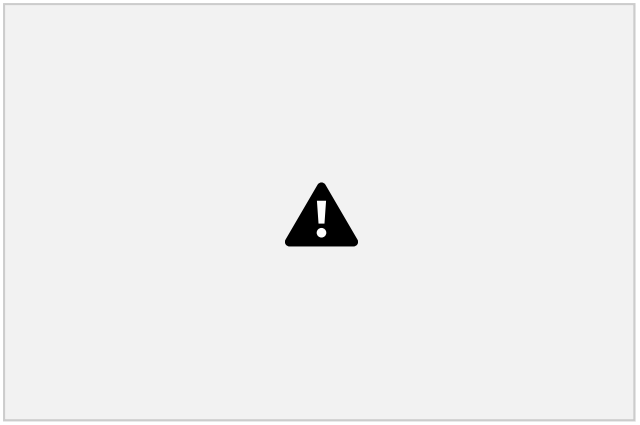
1. StatelessSession

- A StatelessSession does not maintain a first-level cache, nor does it perform dirty checking, associations, or cascading.
- Ideal for bulk inserts, updates, or deletes.
- Use Case: Data shifting/migration

```
SessionFactory sessionFactory =
configuration.buildSessionFactory();
StatelessSession session =
```

```
sessionFactory.openStatelessSession();
```

==========Program: get vs load, FLC & Dirty Checking==========



```
package com.mainapp;
import java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import
```

```
org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import com.entity.Employee;
public class Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
		properties.put(Environment.PASS, "");
		properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
		properties.put(Environment.HBM2DDL_AUTO, "update");
		properties.put(Environment.SHOW_SQL, "true");


		Configuration configuration = new Configuration();
		configuration.setProperties(properties);
		configuration.addAnnotatedClass(Employee.class);

		SessionFactory sessionFactory =
configuration.buildSessionFactory();
```

```
		Session session = sessionFactory.openSession();

		//insert(session);
		 read(session);
```



```
		//update(session);
		// delete(session);

		session.close();
		sessionFactory.close();

	}

	private static void delete(Session session) {

		Transaction transaction = session.getTransaction();

		Employee employee = session.get(Employee.class, 11);

		if(employee!=null) {

			transaction.begin();
			session.delete(employee);
			System.out.println("DATA DELETED");
			transaction.commit();

		}
		else {
			System.out.println("DATA NOT FOUND");
		}
	}
```



```
private static void update(Session
```



```
session) {
```



```
		Transaction transaction = session.getTransaction();
		transaction.begin();
		Employee employee = session.get(Employee.class, 11);

		if(employee!=null) {

			  employee.setEsalary(8976); //DIRTY CHECKING
			System.out.println("DATA UPDATED");
			transaction.commit();

		}else {
			System.out.println("DATA NOT FOUND");
		}

	}

	private static void read(Session session) {
```

```
            Employee employee1 = session.get(Employee.class, 11);
            System.out.println(employee1); //CACHED

            Employee employee2 = session.get(Employee.class, 1676);
            System.out.println(employee2); //CACHED

            //session.clear(); //FREE ALL CACHE
```



```
            session.evict(employee2);

            Employee employee11 = session.get(Employee.class, 11);
            System.out.println(employee11);

            Employee employee22 = session.get(Employee.class, 1676);
            System.out.println(employee22);
      }

      private static void insert(Session session) {

            System.out.println(session);

             Employee employee = new Employee(1676, "name", "address",
1000);
            Transaction transaction = session.getTransaction();

            transaction.begin();

            session.save(employee);

            transaction.commit();
            System.out.println("DATA INSERTED");
      }
}
package com.entity;
import

javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee")
public class Employee {

      @Id
      @Column(name = "id")
      private int eid;

      @Column(name = "name")
      private String ename;
```

```
	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public Employee() {
	}

	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
```

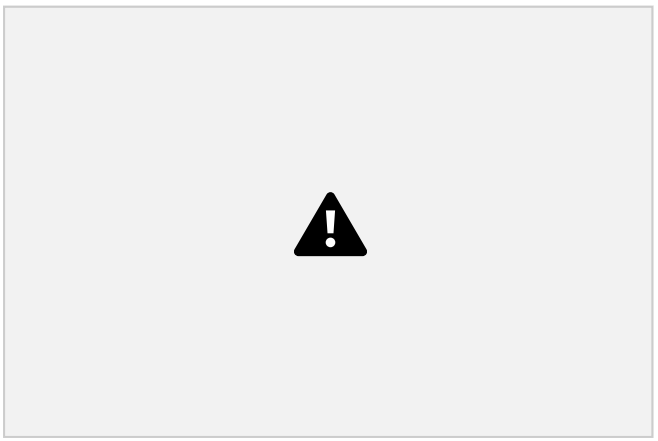


```
		this.ename = ename;
	}

	 public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}
```

```
	public
	String getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	public int getEsalary() {
```

```
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

==========Program: Stateless Session==========



```
package com.mainapp;
import java.util.Properties;
import org.hibernate.SessionFactory;
import org.hibernate.StatelessSession;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import com.entity.Employee;
public class
Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
		properties.put(Environment.PASS, "");
		properties.put(Environment.DRIVER,
```

```
"com.mysql.cj.jdbc.Driver");
            properties.put(Environment.HBM2DDL_AUTO, "update");
            properties.put(Environment.SHOW_SQL, "true");


            Configuration configuration = new Configuration();
            configuration.setProperties(properties);
            configuration.addAnnotatedClass(Employee.class);

            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            StatelessSession session =
sessionFactory.openStatelessSession();

            //insert(session);
            // read(session);
            update(session);
            // delete(session);
```



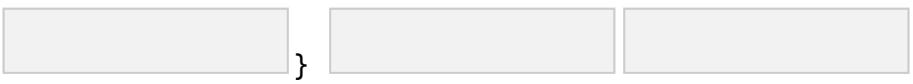
```
            session.close();
            sessionFactory.close();

      }

      private static void delete(StatelessSession session) {

            Transaction transaction = session.getTransaction();

             Employee employee = (Employee) session.get(Employee.class,
11);

            if(employee!=null) {

                  transaction.begin();
                  session.delete(employee);
                  System.out.println("DATA DELETED");
                  transaction.commit();

            }
            else {
                  System.out.println("DATA NOT FOUND");
            }
      }
//
```

```
      update(StatelessSession session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();
             Employee employee = (Employee) session.get(Employee.class,
```

```
11);

            if(employee!=null) {

                  employee.setEsalary(909090);
                  session.update(employee);
                  System.out.println("DATA UPDATED");
                  transaction.commit();

            }else {
                  System.out.println("DATA NOT FOUND");
            }

      }

      private static void read(StatelessSession session) {

             Employee employee1 = (Employee) session.get(Employee.class,
11);
            System.out.println(employee1);

             Employee employee2 = (Employee) session.get(Employee.class,
1676);
```



```
            System.out.println(employee2);

             Employee employee11 = (Employee) session.get(Employee.class,
11);
            System.out.println(employee11);

             Employee employee22 = (Employee) session.get(Employee.class,
1676);
            System.out.println(employee22);
      }

      private static void insert(StatelessSession session) {

            System.out.println(session);

             Employee employee = new Employee(1676, "name", "address",
1000);
           Transaction transaction = session.getTransaction();

            transaction.begin();

            session.insert(employee);

            transaction.commit();
            System.out.println("DATA INSERTED");
      }
```



```
}
```

```java
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee")
public class Employee {

	@Id
	@Column(name = "id")
	private int eid;

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

	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public Employee() {
	}
```



```java
	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
	}

	public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
```

```
	}

	public
	void setEname(String ename) {
		this.ename = ename;
	}

	public String getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	public int getEsalary() {
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

DAY 7:

Bulk Operations in Hibernate Native API

When working with Hibernate Native API, we can perform bulk operations using the following approaches:

Native SQL (createNativeQuery)

- **Allows writing raw SQL queries.**
- **Supports INSERT, UPDATE, and DELETE.**
- **Suitable for performance-critical or database-specific tasks.**

HQL (Hibernate Query Language)

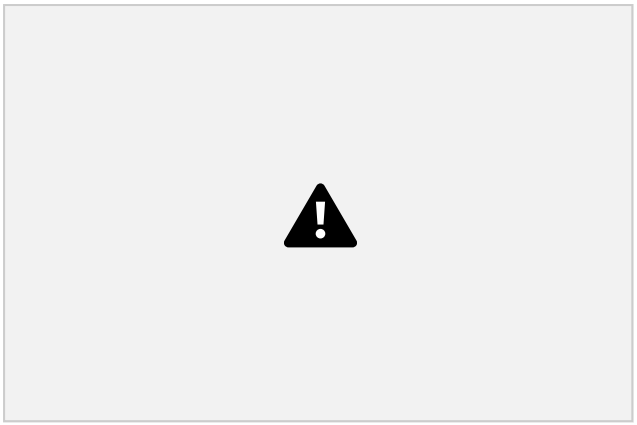
- **Object-oriented query language (uses entity names).**
- **Supports READ,UPDATE and DELETE (bulk).**

- Cannot be used for INSERT operations.
- You can copy data from one table to another (insert into (.....) select)

Criteria API (Deprecated)

- This is the old Hibernate Criteria API (session.createCriteria()).
- Only supports SELECT operations (bulk read).
- Does not support bulk UPDATE, DELETE, or INSERT.
- Deprecated in Hibernate 5, removed in Hibernate 6.

==========Program: Native SQL Bulk Operation==========



```
package com.mainapp;
import java.util.List;
```



```
import java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.query.NativeQuery;
import com.entity.Employee;

public class Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
		properties.put(Environment.PASS, "");
		properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
		properties.put(Environment.HBM2DDL_AUTO, "update");
```

```
            properties.put(Environment.SHOW_SQL, "true");

            Configuration configuration = new Configuration();
            configuration.setProperties(properties);
                                .addAnnotatedClass(E
                                SessionFactory
            sessionFactory =
    configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();

            // insert(session);
             read(session);
            // update(session);
            // delete(session);

            session.close();
            sessionFactory.close();

    }

    private static void delete(Session session) {

        Transaction transaction = session.getTransaction();
        transaction.begin();

        String sql="delete from xemployee where id=:id";
        NativeQuery nativeQuery = session.createNativeQuery(sql);
        nativeQuery.setParameter("id", 1991);
        nativeQuery.executeUpdate();
        System.out.println("DATA DELETED");

        transaction.commit();



    }

    private static void update(Session session) {

        Transaction transaction = session.getTransaction();
        transaction.begin();

        String sql="update xemployee set salary=:salary where id=:id";
        NativeQuery nativeQuery = session.createNativeQuery(sql);
        nativeQuery.setParameter("salary", 987987);
        nativeQuery.setParameter("id", 1991);

        nativeQuery.executeUpdate();
        System.out.println("DATA UPDATED");
        transaction.commit();

    }
```

```
	private static void read(Session session) {

// String sql="select * from xemployee";
// NativeQuery nativeQuery = session.createNativeQuery(sql);
// nativeQuery.addEntity(Employee.class);
// List<Employee> list = nativeQuery.getResultList();
// for(Employee emp: list)
// System.out.println(emp);
```



```
NativeQuery nativeQuery =
session.getNamedNativeQuery("readAll");
		nativeQuery.addEntity(Employee.class);
		List<Employee> list = nativeQuery.getResultList();
		for(Employee emp: list)
			System.out.println(emp);

	}

	private static void insert(Session session) {

		Transaction transaction = session.getTransaction();
		transaction.begin();

		 String sql="insert into xemployee(id,name,address,salary)
values (:id,:name,:address,:salary)";
		NativeQuery nativeQuery = session.createNativeQuery(sql);
		nativeQuery.setParameter("id", 1991);
		nativeQuery.setParameter("name", "namex");
		nativeQuery.setParameter("address", "addressx");
		nativeQuery.setParameter("salary", 123456);

		nativeQuery.executeUpdate();
		System.out.println("DATA INSERTED");
		transaction.commit();
	}
```

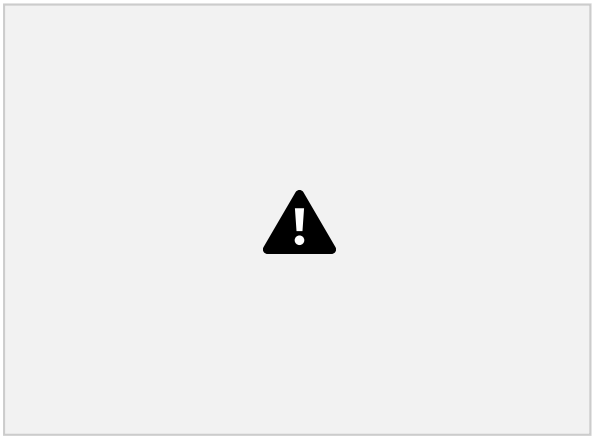
```
}
```

```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedNativeQuery;
import javax.persistence.Table;

@Entity
@Table(name = "xemployee")
@NamedNativeQuery(name = "readAll",query = "select * from xemployee")
public class Employee {

	@Id
	@Column(name = "id")
	private int eid;
```

```
	@Column(name = "name")
	private String ename;

	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public Employee() {
	}

	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
	}

	public Employee(int eid, String ename, String eaddress, int esalary)
{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public String getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
```

```
	}

	public int getEsalary() {
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		return "Employee [eid=" + eid + ", ename=" + ename + ", eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

======Program: Native SQL Bulk Operation========



```
package com.mainapp;
import java.util.List;
import java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.query.Query;
```

```
import com.entity.Employee;
import com.entity.EmployeeTest;
public class Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL, "jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
```

```
            properties.put(Environment.PASS, "");
            properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
            properties.put(Environment.HBM2DDL_AUTO, "update");
            properties.put(Environment.SHOW_SQL, "true");

            Configuration configuration = new Configuration();
            configuration.setProperties(properties);
            configuration.addAnnotatedClass(Employee.class);
            configuration.addAnnotatedClass(EmployeeTest.class);

            SessionFactory sessionFactory =
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();

            // insert(session);
                                    read(session);
                                    // update(session);
                                    // delete(session);

            session.close();
            sessionFactory.close();

      }

      private static void delete(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();

            String hql="delete from Employee where eid=:eid";
            Query query = session.createQuery(hql);
            query.setParameter("eid", 11);
            query.executeUpdate();
            System.out.println("DATA DELETED");

            transaction.commit();

      }

      private static void update(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();
```



```
             String hql="update Employee set esalary=:esalary where
eid=:eid";
            Query query = session.createQuery(hql);
            query.setParameter("esalary", 8080);
            query.setParameter("eid", 11);

            query.executeUpdate();
```

```
            System.out.println("DATA UPDATED");
            transaction.commit();

      }

      private static void read(Session session) {

// String hql="from Employee";
// Query query = session.createQuery(hql);
// List<Employee> list = query.list();
// for(Employee e : list) {
// System.out.println(e);
// }

            Query query = session.getNamedQuery("readAll");
            List<Employee> list = query.list();
            for(Employee e : list) {
                  System.out.println(e);

            }
      }

      //copy
      private static void insert(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();

            String hql="insert into
EmployeeTest(eid,ename,eaddress,esalary) select eid,ename,eaddress,esalary
from Employee";
            Query query = session.createQuery(hql);
            query.executeUpdate();
            System.out.println("DATA COPIED....");
            transaction.commit();
      }
}


package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.Table;




@Entity
@Table(name = "xemployee")
@NamedQuery(name = "readAll" , query = "from Employee")
```

```
public class Employee {

	@Id
	@Column(name = "id")
	private int eid;

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

	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public Employee() {
	}

	public Employee(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
	}

public Employee(int eid, String ename,
                String eaddress, int esalary)
	{
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public String getEaddress() {
		return eaddress;
```

```
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	public int getEsalary() {
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import
```

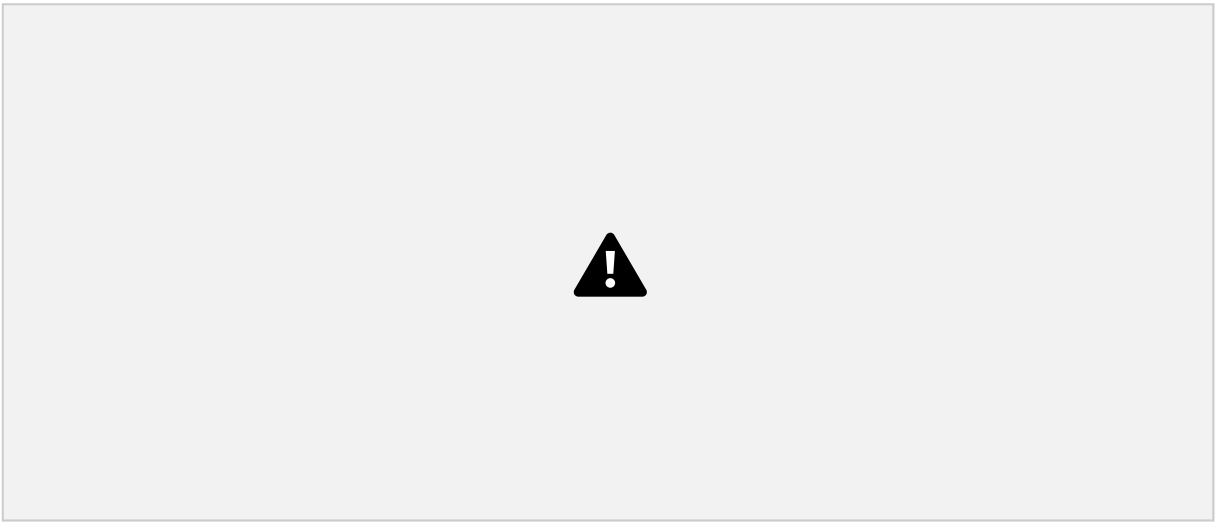
```
javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "xemployeetest")

public class EmployeeTest {

	@Id
	@Column(name = "id")
	private int eid;

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

	@Column(name = "address")
	private String eaddress;

	@Column(name = "salary")
	private int esalary;

	public EmployeeTest() {
	}

	public EmployeeTest(int eid, String ename) {
		super();
		this.eid = eid;
		this.ename = ename;
```

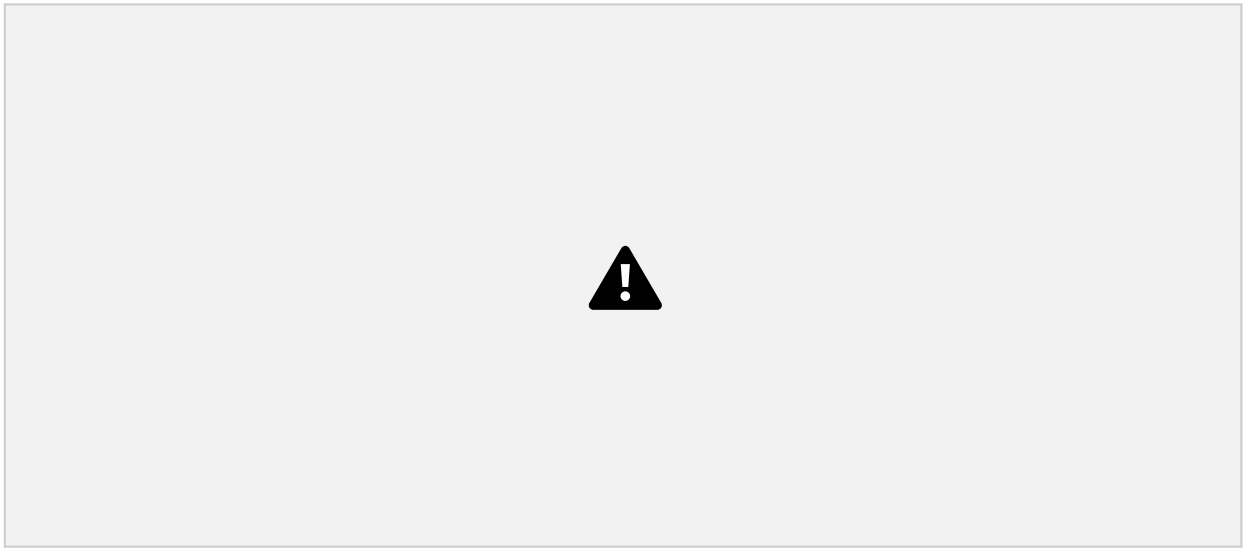
```
	}

	public EmployeeTest(int eid, String ename, String eaddress, int
esalary) {
		super();
		this.eid = eid;
		this.ename = ename;
		this.eaddress = eaddress;
		this.esalary = esalary;
	}

	public int getEid() {
		return eid;
	}

	public void setEid(int eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public String
	getEaddress() {
		return eaddress;
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	public int getEsalary() {
		return esalary;
	}

	public void setEsalary(int esalary) {
		this.esalary = esalary;
	}

	@Override
	public String toString() {
		 return "Employee [eid=" + eid + ", ename=" + ename + ",
eaddress=" + eaddress + ", esalary=" + esalary + "]";
	}
}
```

DAY 8:

Hibernate Inheritance

- In Hibernate and JPA, you can use inheritance to let one entity class extend another (like Car extends Vehicle).
- Hibernate and JPA support this by allowing you to map inheritance to database tables in different ways.
- When you write an HQL or JPQL query using the parent class, it will also return objects of its child classes and This is called polymorphic behaviour





Inheritance Strategies in Hibernate (HQL)

Hibernate supports three inheritance mapping strategies:

1. Single Table (Default)

- All classes in the hierarchy are mapped to one table.
  - A discriminator column is used to differentiate between entity types.
- Efficient (no joins) but may have many nullable columns.

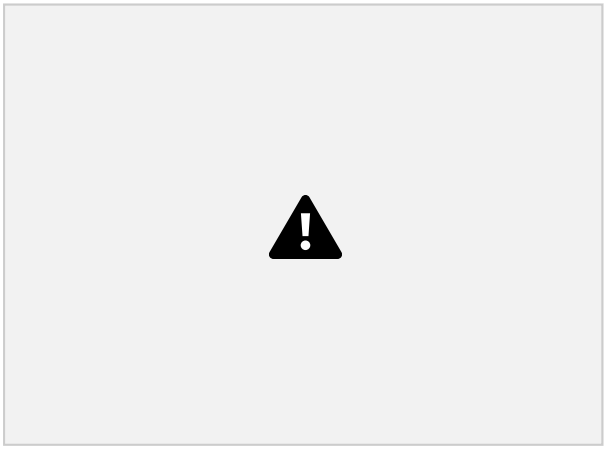
2. Table Per Class

- Each class has its own separate table.
- All inherited fields are duplicated in subclass tables.
- Hibernate uses UNIONs to query parent class.

3. Joined Strategy

- Parent and child entities are stored in separate tables, joined by primary key.
- Most normalized design; fewer nulls.

==========Program: Hibernate Inheritance==========



```
package com.mainapp;
import java.util.List;
import
```



```
java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
```

```
import org.hibernate.query.NativeQuery;
import org.hibernate.query.Query;
import com.entity.Player;
import com.entity.Cricketer;
import com.entity.Footballer;
public class Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
		properties.put(Environment.PASS, "");
		properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
		properties.put(Environment.HBM2DDL_AUTO, "update");
		properties.put(Environment.SHOW_SQL, "true");

		Configuration configuration = new Configuration();
		configuration.setProperties(properties);
		configuration.addAnnotatedClass(Player.class);
```



```
		configuration.addAnnotatedClass(Cricketer.class);
		configuration.addAnnotatedClass(Footballer.class);

		SessionFactory sessionFactory =
configuration.buildSessionFactory();
		Session session = sessionFactory.openSession();

	 //insert(session);
		//readSQL(session);
		readHQL(session);
		// update(session);
		// delete(session);

		session.close();
		sessionFactory.close();

	}

	private static void readSQL(Session session) {

		 String sql="select * from cricketer UNION ALL select * from
Footballer";
		NativeQuery nativeQuery = session.createNativeQuery(sql);
		List<Object[]> list = nativeQuery.getResultList();
		for(Object[] orr : list) {
			for(Object o : orr) {
```



```
				System.out.print(o+" ");
```

```
                }
                System.out.println();
            }

        }

        //POLYMORPHIC QUERY
    private static void readHQL(Session session) {

    String hql="from Player";
    Query query = session.createQuery(hql);
    List<Player> list = query.list();
    for( Player p:list) {
    if(p instanceof Cricketer) {
    System.out.println(p);
    }else {
    System.out.println(p);
    }
    }

        }

        private static void insert(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();
```



```
            Cricketer cricketer = new Cricketer(123, "raju", 1000, "wk");
            Footballer footballer = new Footballer(125, "kaju", 70, "gk");

            session.save(cricketer);
            session.save(footballer);

            transaction.commit();
        }
}
```

```
package com.entity;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

//@Entity
//@Inheritance(strategy = InheritanceType.JOINED)


//@Entity
//@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
```

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)

//OR
//@MappedSuperclass
//@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Player {

	@Id
	private int id;
	private String name;

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

	public Player(int id, String name) {
		super();
		this.id = id;
		this.name = name;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
```



```java
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@Override
	public String toString() {
		return "Player [id=" + id + ", name=" + name + "]";
	}
}
```

```java
package com.entity;
import javax.persistence.Entity;
```

```java
@Entity
public class Footballer extends Player {

	private int goal;
	private String ftype;

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

	public Footballer(int id, String name, int goal, String ftype) {
		super(id, name);
		this.goal = goal;
		this.ftype = ftype;
	}

	public int getGoal() {
		return goal;
	}

	public void setGoal(int goal) {
		this.goal = goal;
	}

	public String getFtype() {
		return ftype;
	}

	public void setFtype(String ftype) {
		this.ftype = ftype;
	}

	@Override
	public String toString() {
		return "Footballer [goal=" + goal + ", ftype=" + ftype + "]";
	}
}
```

```java
package com.entity;
import javax.persistence.Entity;

@Entity
public class Cricketer extends Player {

	private int run;
	private String ctype;

	public Cricketer() {
```

```
		// TODO Auto-generated constructor stub
	}

	public Cricketer(int id, String name, int run, String ctype) {
		super(id, name);
		this.run = run;
		this.ctype = ctype;
	}

	public
	int getRun() {
		return run;
	}

	public void setRun(int run) {
		this.run = run;
	}

	public String getCtype() {
		return ctype;
	}

	public void setCtype(String ctype) {
		this.ctype = ctype;
	}

	@Override
	public String toString() {
		return "Cricketer [run=" + run + ", ctype=" + ctype + "]";
	}
}
```

DAY 9:

Embeddable and @Embedded

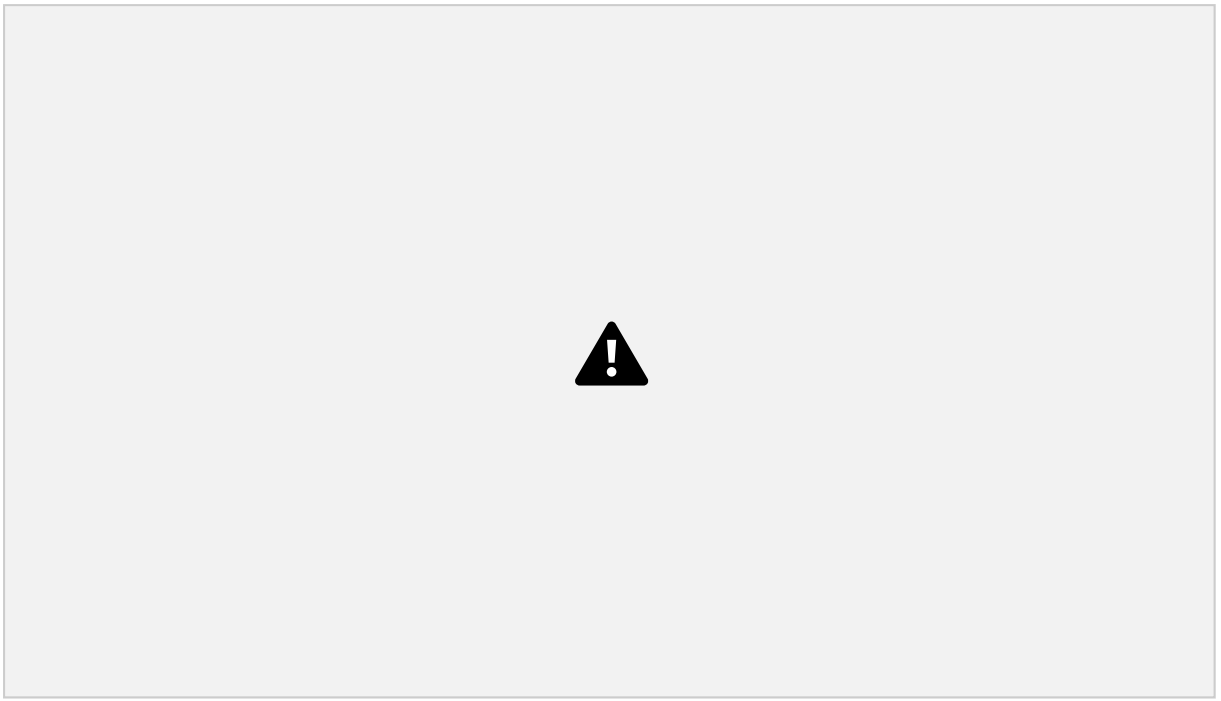
- **In Hibernate and JPA, @Embeddable is used to mark a class whose fields can be embedded into another entity. These classes represent**

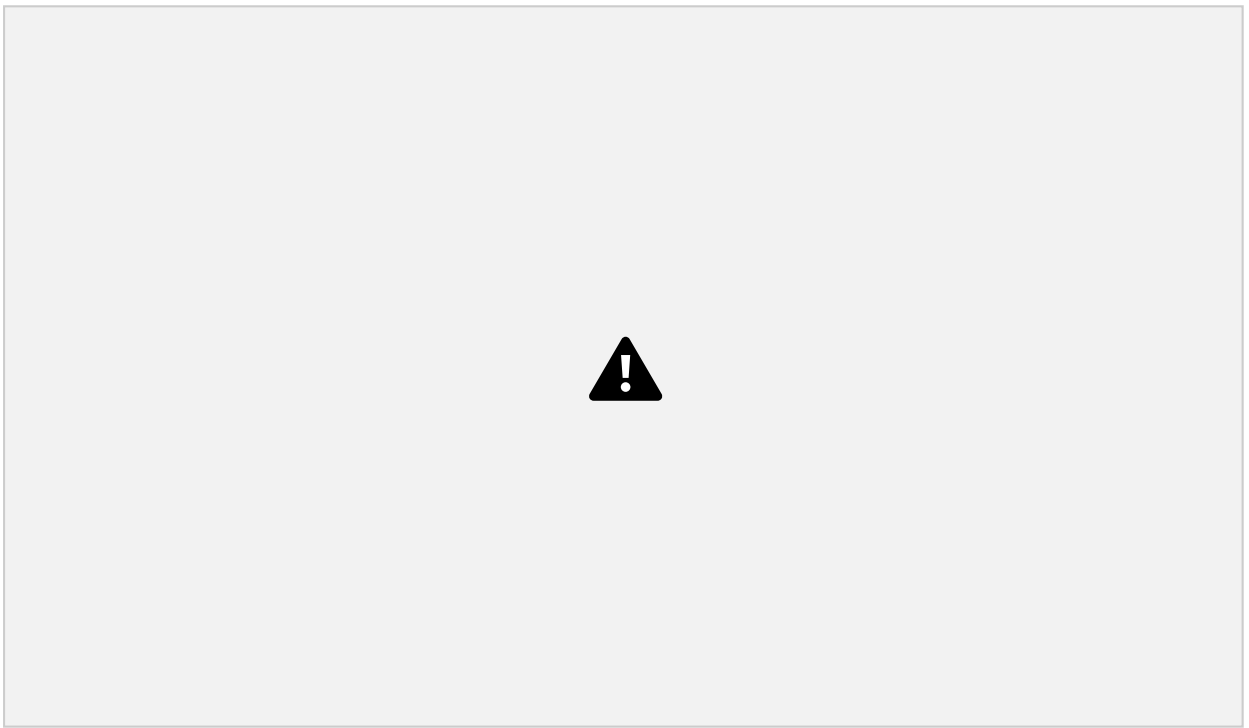
reusable components or value objects like Address, Car, or Account, and they do not have their own primary key. You can think of them as pieces of information that are part of another entity, not standalone entities themselves.

- **The @Embedded annotation is used inside an entity to include an object of an @Embeddable class. When an embeddable object is used in an entity with @Embedded, all its fields are mapped as columns in the same table as the entity. This helps in better normalization and reusability of logic while keeping the database structure flat.**

@Version

- The @Version annotation is used to enable optimistic locking in Hibernate. It introduces a version column in the entity's table that keeps track of how many times a row has been updated.





========Program: Hibernate @Embedded @Embeddable @Version======

```
package com.mainapp;
import java.time.LocalDateTime;
import java.util.Properties;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import
```

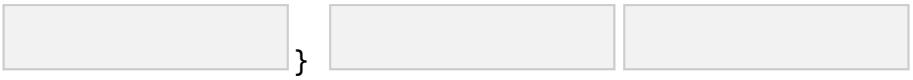


```
com.entity.Account;
import com.entity.Car;
import com.entity.Company;
import com.entity.Employee;
public class Launch {

	public static void main(String[] args) {

		Properties properties = new Properties();
		properties.put(Environment.URL,
"jdbc:mysql://localhost:3306/hiber");
		properties.put(Environment.USER, "root");
		properties.put(Environment.PASS, "");
		properties.put(Environment.DRIVER,
"com.mysql.cj.jdbc.Driver");
		properties.put(Environment.HBM2DDL_AUTO, "update");
		properties.put(Environment.SHOW_SQL, "true");

		Configuration configuration = new Configuration();
		configuration.setProperties(properties);
		configuration.addAnnotatedClass(Employee.class);

		SessionFactory sessionFactory =
```

```
configuration.buildSessionFactory();
            Session session = sessionFactory.openSession();

       //insert(session);
            //read(session);
```



```
            update(session);
            //delete(session);

            session.close();
            sessionFactory.close();

      }

      public static void update(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();

            Employee employee = session.get(Employee.class, "eid12343");
            employee.setEmployeeName("KAJU");

            session.update(employee);
            transaction.commit();
            System.out.println("DATA UPDATED");

      }

      private static void read(Session session) {

            Employee employee = session.get(Employee.class, "eid12343");
            System.out.println(employee);
```



```
      }

      private static void insert(Session session) {

            Transaction transaction = session.getTransaction();
            transaction.begin();

             Account account = new Account("1212ABCD", "RAJUACC",
"RAJUACCADDR", "XYZBANK", "ABCD0qwerty");
            Car car = new Car("UP671234",
"XYZCARMODEL",LocalDateTime.now() , 1000.0);

             Company company = new Company("reg1234", "xyzcnname",
"policy", account);
             Employee employee = new Employee("eid12343", "RAJU", company,
car);

            session.save(employee);
```

```
            transaction.commit();
      }
}
```

```
package com.entity;
import javax.persistence.Embedded;
```



```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Version;

@Entity
public class Employee {

      @Id
      private String employeeId;
      private String employeeName;

      @Embedded
      private Company company;

      @Embedded
      private Car car;

      @Version
      private int version;

      public Employee() {

      }

       public Employee(String employeeId, String employeeName, Company
company, Car car) {
            super();
            this.employeeId =
            employeeId;
            this.employeeName = employeeName;
            this.company = company;
            this.car = car;
      }

      public String getEmployeeId() {
            return employeeId;
      }

      public void setEmployeeId(String employeeId) {
            this.employeeId = employeeId;
      }

      public String getEmployeeName() {
```

```
		return employeeName;
	}

	public void setEmployeeName(String employeeName) {
		this.employeeName = employeeName;
	}

	public Company getCompany() {
		return company;
	}

	public void setCompany(Company company) {
```



```
		this.company = company;
	}

	public Car getCar() {
		return car;
	}

	public void setCar(Car car) {
		this.car = car;
	}

	@Override
	public String toString() {
		return "Employee [employeeId=" + employeeId + ",
employeeName=" + employeeName + ", company=" + company
				+ ", car=" + car + "]";
	}

}


package com.entity;
import javax.persistence.Embeddable;
import javax.persistence.Embedded;

@Embeddable
public class
Company {

	private String registrationNo;
	private String companyName;
	private String policy;

	@Embedded
	private Account account;

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

```
	public Company(String registrationNo, String companyName, String
policy, Account account) {
		super();
		this.registrationNo = registrationNo;
		this.companyName = companyName;
		this.policy = policy;
		this.account = account;
	}

	public String getRegistrationNo() {
		return registrationNo;
	}

	public void setRegistrationNo(String registrationNo) {
```