Voici un exemple pratique d'utilisation de **JPA** avec **MySQL** comme base de données.

---

1. Ajouter les dépendances dans Maven

Si vous utilisez Maven, ajoutez les dépendances suivantes dans votre fichier `pom.xml` :

```
<dependencies>
    <!-- JPA API -->
    <dependency>
        <groupId>jakarta.persistence</groupId>
        <artifactId>jakarta.persistence-api</artifactId>
        <version>3.1.0</version>
    </dependency>

    <!-- Hibernate comme fournisseur JPA -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.2.6.Final</version>
    </dependency>

    <!-- Connecteur JDBC pour MySQL -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.1.0</version>
    </dependency>
</dependencies>
```

---

2. Configuration de JPA avec MySQL

Créez le fichier `persistence.xml` dans le dossier `src/main/resources/META-INF` et configurez-le comme suit :

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
jakarta-persistence_3_1.xsd"
             version="3.1">
    <persistence-unit name="examplePU" transaction-type="RESOURCE_LOCAL">
        <class>com.example.model.Person</class>
        <properties>
            <!-- Informations sur la base de données -->
            <property name="jakarta.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/testdb"/>
            <property name="jakarta.persistence.jdbc.user" value="root"/>
            <property name="jakarta.persistence.jdbc.password"
value="password"/>
            <property name="jakarta.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver"/>

            <!-- Hibernate en tant que fournisseur JPA -->
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
```

```
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

3. Exemple d'entité JPA

Voici une classe `Person` mappée à une table dans MySQL :

```
package com.example.model;

import jakarta.persistence.*;

@Entity
@Table(name = "person") // Le nom de la table dans MySQL
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-
incrémentation
    private Long id;

    @Column(nullable = false) // Champ obligatoire
    private String name;

    @Column
    private int age;

    // Getters et Setters
    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

---

4. Code Principal pour Persister les Données

Voici un programme principal pour insérer et lire des données de la table `person`.

```
package com.example;

import com.example.model.Person;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class Main {
    public static void main(String[] args) {
        // Créer l'EntityManagerFactory à partir de persistence.xml
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("examplePU");
        EntityManager em = emf.createEntityManager();

        // Début de la transaction
        em.getTransaction().begin();

        // Créer et persister une nouvelle personne
        Person person = new Person();
        person.setName("Alice");
        person.setAge(30);
        em.persist(person);

        // Fin de la transaction
        em.getTransaction().commit();

        // Récupérer et afficher la personne par son ID
        Person foundPerson = em.find(Person.class, person.getId());
        System.out.println("Found Person: " + foundPerson.getName() + ",
Age: " + foundPerson.getAge());

        // Fermer l'EntityManager et l'EntityManagerFactory
        em.close();
        emf.close();
    }
}
```

---

5. Configuration de MySQL

1. **Créer une base de données** : Exécutez cette commande SQL pour créer une base de données :
2. `CREATE DATABASE testdb;`
3. **Configurer un utilisateur MySQL** : Vérifiez que l'utilisateur `root` avec le mot de passe `password` est configuré pour accéder à la base de données.
4. **Vérifiez la table générée** : Une fois que vous exécutez le programme, Hibernate créera automatiquement la table `person` grâce à l'option `hibernate.hbm2ddl.auto=update`.

---

6. Fonctionnalités supplémentaires

Lire plusieurs enregistrements :

```
List<Person> people = em.createQuery("SELECT p FROM Person p",
Person.class).getResultList();
```

```
for (Person p : people) {
    System.out.println("Person: " + p.getName() + ", Age: " + p.getAge());
}
```

Supprimer une entité :

```
em.getTransaction().begin();
Person personToDelete = em.find(Person.class, personId);
if (personToDelete != null) {
    em.remove(personToDelete);
}
em.getTransaction().commit();
```

Mettre à jour une entité :

```
em.getTransaction().begin();
Person personToUpdate = em.find(Person.class, personId);
if (personToUpdate != null) {
    personToUpdate.setAge(35);
}
em.getTransaction().commit();
```

---

7. Tester l'Application

1. Lancez le programme.
2. Vérifiez dans votre base de données MySQL que les données sont insérées dans la table `person`.

Résumé

1. **JPA** simplifie les interactions avec une base de données.
2. **Hibernate** est souvent utilisé comme implémentation de JPA.
3. La configuration est définie dans `persistence.xml`.
4. Les opérations CRUD peuvent être réalisées directement via l'`EntityManager`.