CS 513 Software Systems(ESD)

Lab - 3 Hibernate and JPA

Object-Relational Mapping (ORM) is the process of converting Java objects to database tables. In other words, this allows us to interact with a relational database without any SQL. **The Java Persistence API (JPA) is a specification that defines how to persist data in Java applications.** The primary focus of JPA is the ORM layer.

JDBC:

JDBC stands for **Java Database Connectivity**. It is a java application programming interface to provide a connection between the Java programming language and a wide range of databases (i.e), it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

HIBERNATE :

Hibernate is a high-performance open source Object-Relational Mapping framework and query service. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities.

DIFFERENCE:

The short answer on the fundamental difference between JDBC and Hibernate is that Hibernate performs an object-relational mapping framework, while JDBC is simply a database connectivity API.

Hibernate is one of the most popular Java ORM frameworks in use today.

1. Hibernate supports the mapping of java classes to database tables and vice versa. It provides features to perform CRUD operations across all the major relational

databases.

2. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business use cases rather than making sure that database operations are not causing resource leaks.
3. Hibernate supports transaction management and make sure there is no inconsistent data present in the system.

SETUP:

Pre-Requisite: MySQL installed.

Create a Maven Project::

Hibernate Project Dependencies:

In **pom.xml: Add the following dependencies**

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.5.3.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.22</version>
    </dependency>
</dependencies>
```

hibernate-core artifact contains all the core hibernate classes, so we will get all the necessary features by including it in the project.

mysql-connector-java is the MySQL driver for connecting to MySQL databases, if you are using any other database then add corresponding driver artifact.

Hibernate Configuration Files

Under **src/main/resources**, create:

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:mysql://localhost:3306/hibernatelab?createDat
abaseIfNotExist=true</property>
        <property name="connection.username">root</property>
        <property name="connection.password">password</property>
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>

        <!-- DB schema will be updated if needed -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="show_sql">true</property>
        <mapping class="com.example.erp.bean.Employees"/>
        <mapping class="com.example.erp.bean.Departments"/>
        <mapping class="com.example.erp.bean.Projects"/>
    </session-factory>

</hibernate-configuration>
```

https://github.com/neha-kothari/esd-hibernate-lab/blob/main/src/main/resources/hibernate.cfg.xml

Hibernate SessionFactory

https://github.com/neha-kothari/esd-hibernate-lab/blob/main/src/main/java/com/example/erp/util/HibernateSessionUtil.java

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The

org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

HIBERNATE ANNOTATIONS:

@Entity annotation marks this class as an entity.

@Table annotation specifies the table name where data of this entity is to be persisted. If you don't use @Table annotation, hibernate will use the class name as the table name by default.

@Id annotation marks the identifier for this entity.

@Column annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.

@GeneratedValue annotation specifies how to generate values for the given column. This annotation will help in creating primary keys values according to the specified strategy. https://www.tutorialandexample.com/hibernate-generatedvalue-strategies/

Eager and Lazy Loading

- **Eager Loading** is a design pattern in which data initialization occurs on the spot
- **Lazy Loading** is a design pattern which is used to defer initialization of an object as long as it's possible

Useful Links:

https://github.com/neha-kothari/esd-hibernate-lab/tree/main

https://www.journaldev.com/17803/jpa-hibernate-annotations