Использование Hibernate для манипулирования данными в базе данных с помощью Maven.

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения. Позволяет сократить объёмы низкоуровневого программирования при работе с реляционными базами данных; может использоваться как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой.

Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL.

1. Создаём проект с помощью maven.

Как зависимости подключаем jdbc от postgresql и hibernate:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>6.0.0.CR1</version>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.3.3</version>
    </dependency>
</dependencies>
```

Все зависимости можно найти в репозитории Maven:
https://mvnrepository.com/

2. Создаём таблицу Client в базе данных:

```
denis@/run/postgresql:BankDB> create table client(
 id_client smallint primary key,
 name varchar(30) not null,
 surname varchar(30) not null,
 date_birth date not null,
 telephone integer
 );
CREATE TABLE
Time: 0.010s
denis@/run/postgresql:BankDB>
```

и вводим туда запись:

```
denis@/run/postgresql:BankDB> select * from client;
+-----------+------+---------+------------+-----------+
| id_client | name | surname | date_birth | telephone |
|-----------+------+---------+------------+-----------|
| 1         | Igor | Verlan  | 2022-02-01 | 68618833  |
+-----------+------+---------+------------+-----------+
```

3. В resources создаём файл hibernate.cfg.xml:
Для работы hibernet мы должны имеет базовый конфигурационный файл hibernate.cfg.xml, он должен находиться в корне папки ресурсов.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="http://www.hibernate.org/xsd/orm/cfg">
    <session-factory>
        <property name="connection.url">jdbc:postgresql://192.168.43.184/BankDB</property>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">1111</property>
        <property name="dialect">org.hibernate.dialect.PostgreSQL91Dialect</property>
        <property name="show_sql">true</property>


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

В нём есть:
1. Путь к базе данных

```
<property name="connection.url">jdbc:postgresql://192.168.43.184/BankDB</property>
```

2. Класс работы с jdbc драйвером

```
<property name="connection.driver_class">org.postgresql.Driver</property>
```

3. Имя пользователя

```
<property name="connection.username">postgres</property>
```

4. Пороль

```
<property name="connection.password">1111</property>
```

5. Диалект

```
<property name="dialect">org.hibernate.dialect.PostgreSQL91Dialect</property>
```

6. Путь отладки, для вывода запросов на sql в терминале

```
<property name="show_sql">true</property>
```

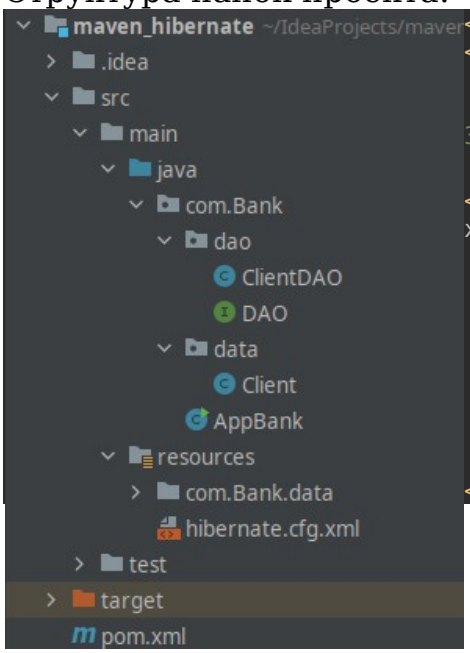
Тегом mapping, мы ссылаемся на файл который будет описывать то как соотносятся наши поля в таблице с нашим классом. Для каждого класса, который представляет таблицу должен быть создан такой файл.

```
<mapping resource="com/Bank/data/Client.hbm.xml" />
```

3. Создаём файл с названием нашей таблицы в БД «Client.hbm.xml», в директории, которая в точности повторяет путь до нашего класса Client.

Это конфигурационный файл hibernate, должен называться точно также как и класс, к которому он соотносится, должен повторять структуру, которая есть в папке java, до файла класса.

Структура папок проекта:



Client.hbm.xml:

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

<hibernate-mapping
xmlns="http://www.hibernate.org/xsd/hibernate-mapping">

    <class name="com.Bank.data.Client" table="client">
        <id name="idClient" column="id_client"/>
        <property name="name" column="name"/>
        <property name="surname" column="surname"/>
        <property name="dateBirth" column="date_birth"/>
        <property name="telephone" column="telephone"/>
    </class>
</hibernate-mapping>
```

У нас есть главный тег класс в котором мы указываем путь к классу, и привязку к таблице. Тег id — уникальный ключ, представляется именем поля в классе и столбцом в таблице. Тег proprty включает все остальные столбцы, представление такое же.

4. Создаём класс Client, для хронения информации о клиенте:

Для того, что бы связать java объект и нашу таблицу он должен соответствовать ряду требований:
1. У объекта должен быть пустой конструктор.
2. Должны быть геттеры и сеттеры.
3. Класс не должен быть финальным.
У таблицы должен быть уникальный ID

```
package com.Bank.data;

import java.sql.*;

public class Client {

    private int idClient;
    private String name;
    private String surname;
    private Date dateBirth;
    private int telephone;
    @Override
```

```
    public String toString() {
        return "Client{" +
                "idClient=" + idClient +
                ", name='" + name + '\'' +
                ", surname='" + surname + '\'' +
                ", dateBirth=" + dateBirth +
                ", telephone=" + telephone +
                '}';
    }
    public int getIdClient() {
        return idClient;
    }
    public String getName() {
        return name;
    }
    public String getSurname() {
        return surname;
    }
    public Date getDateBirth() {
        return dateBirth;
    }
    public int getTelephone() {
        return telephone;
    }
    public void setIdClient(int idClient) {
        this.idClient = idClient;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public void setDateBirth(Date dateBirth) {
        this.dateBirth = dateBirth;
    }
    public void setTelephone(int telephone) {
        this.telephone = telephone;
    }
}
```

5. Создаём интерфейс «DAO» и класс «ClientDAO» для работы с классом «Client».

DAO:

```
package com.Bank.dao;

public interface DAO<Entity, Key> {
    void create(Entity entity);
    Entity read(Key key);
    void update(Entity entity);
    void delete(Entity entity);
}
```

ClientDAO:

```
package com.Bank.dao;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import com.Bank.data.Client;
```

```java
public class ClientDAO implements DAO<Client,Integer>{
    private final SessionFactory factory;

    public ClientDAO(final SessionFactory factory) {
        this.factory = factory;
    }

    @Override
    public void create(final Client client) {
        try (final Session session = factory.openSession()) {
            session.beginTransaction();
            session.save(client);
            session.getTransaction().commit();
        }
    }

    @Override
    public Client read(final Integer idClient) {
        try (final Session session = factory.openSession()) {
            final Client result = session.get(Client.class, idClient);
            return result != null ? result : new Client();
        }
    }

    @Override
    public void update(final Client client) {
        try (Session session = factory.openSession()) {
            session.beginTransaction();
            session.update(client);
            session.getTransaction().commit();
        }
    }

    @Override
    public void delete(final Client client) {
        try (Session session = factory.openSession()) {
            session.beginTransaction();
            session.delete(client);
            session.getTransaction().commit();
        }
    }
}
```

Метод create:

1. Открываем сессию Session session = factory.openSession() в try

```java
final Session session = factory.openSession()
```

Интерфейс org.hibernate.Session является мостом между приложением и Hibernate. С помощью сессий выполняются все CRUD-операции с объектами-сущностями. Объект типа Session получают из экземпляра типа org.hibernate.SessionFactory, который должен присутствовать в приложении в виде singleton.

2. Начинаем транзакцию

```java
session.beginTransaction();
```

3. сохраняем объект

```java
session.save(client);
```

4. Получаем транзакцию и делаем коммит

```java
session.getTransaction().commit();
```

read:
1. Открываем сессию
```
final Session session = factory.openSession()
```
2. вызываем метод get посылаем туда класс, который должен быть сгенерирован и первичный ключ по которому будет сделана выборка.
```
final Client result = session.get(Client.class, idClient);
```
3. Если не равно null, возвращаю объект, иначе создаю объект и возвращаю его.
```
return result != null ? result : new Client();
```

Update:
1. Открываем сессию
```
Session session = factory.openSession()
```
2. Начинает транзакцию
```
session.beginTransaction();
```
3. вызываем метод update и обновляем строку в таблице
```
session.update(client);
```
4. Получаем транзакцию и делаем коммит
```
session.getTransaction().commit();
```

Delete:
1. Открываем сессию
```
Session session = factory.openSession()
```
2. Начинает транзакцию
```
session.beginTransaction();
```
3. вызываем метод delete и удаляем строку в таблице
```
session.delete(client);
```
4. Получаем транзакцию и делаем коммит
```
session.getTransaction().commit();
```

6. В классе AppBank прописываем следующий код:

```
package com.Bank;

import com.Bank.data.Client;
import com.Bank.dao.DAO;
import com.Bank.dao.ClientDAO;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import java.sql.Date;

public class AppBank {
        public static void main(String[] args) {
                SessionFactory factory = null;
                try {
                        factory = new Configuration().configure().buildSessionFactory();
                        DAO<Client, Integer> clientDAO = new ClientDAO(factory);

                        /**Ввод нового пользователя в таблицу**/
                        Client client = new Client();
                        client.setIdClient(2);
                        client.setName("Andrei");
                        client.setSurname("Testemitianu");
                        client.setDateBirth(Date.valueOf("2022-01-11"));
                        client.setTelephone(76560012);
```

```
                        clientDAO.create(client);

                } finally {
                        if (factory != null) {
                                factory.close();
                        }
                }
        }
}
```

Для работы с hibernate создаём объект SessionFactory.
Для того, что бы получить объект мы используем класс Configuration().
вызываем метод configure() и buildSessionFactory()

Запускаем программу и проверяем нашу таблицу через СУБД

```
BankDB=# select * from Client;
 id_client |  name  |   surname    | date_birth | telephone
-----------+--------+--------------+------------+-----------
         1 | Igor   | Verlan       | 1997-01-23 |  68618833
         2 | Andrei | Testemitianu | 2022-01-11 |  76560012
(2 строки)
```

Ввод данных успешен.
Мы проверили ввод клиента в таблицу. Заменим код в try от комментария до finally на:

```
/**Чтение из БД пользователя по id**/
Client result = clientDAO.read(2);
System.out.println("Read : " + result);
System.out.println();
```

Запускаем. Получаем результат:

```
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate: select cl_0.id_client,cl_0.date_birth,cl_0.name,cl_0.surname,cl_0.telephone from client cl_0 where cl_0.id_client=?
Read : Client{idClient=2, name='Andrei', surname='Testemitianu', dateBirth=2022-01-11, telephone=76560012}
```

Чтение прошло успешно.
Далее обновим данные о клиенте в таблице. Для этого заменим код на новый:

```
/**Обновление данных о пользователе**/
Client result = clientDAO.read(2);
result.setTelephone(65004515);
clientDAO.update(result);
```

Запускаем программу и проверяем изменились ли данные о клиенте в СУБД:

```
BankDB=# select * from Client;
 id_client |  name  |   surname    | date_birth | telephone
-----------+--------+--------------+------------+-----------
         1 | Igor   | Verlan       | 1997-01-23 |  68618833
         2 | Andrei | Testemitianu | 2022-01-11 |  65004515
(2 строки)
```

Можем сравнить данные до и после. Операция прошла успешно.

Последний метод, метод удаления клиента. Вставим следующий код:

```
/**Удаление клиента из таблицы**/
Client client = clientDAO.read(2);
clientDAO.delete(client);
```

Запускаем, проверяем работу:

```
BankDB=# select * from Client;
 id_client | name | surname | date_birth | telephone
-----------+------+---------+------------+-----------
         1 | Igor | Verlan  | 1997-01-23 |  68618833
(1 строка)
```

Запись о клиенте с id равное 2 отсутствует.