

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

## Многие записи к одной.

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 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  |
+-----+-----+-----+-----+-----+
```

Следом создаём вторую таблицу bank\_card, у которой есть внешний ключ на таблицу Client:

```
BankDB=# create table bank_card (
id_card smallint primary key,
type varchar(20) not null,
date_issue date not null,
id_client smallint not null,
constraint fk_id_client
foreign key (id_client)
references client(id_client)
);
```

и вводим туда запись:

```
id_card | type   | date_issue | id_client
-----+-----+-----+-----
1       | social | 2022-02-01 | 1
```

3. В resources создаём файл hibernate.cfg.xml:

Для работы hibernate мы должны иметь 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" />
    <mapping resource="com/Bank/data/Card.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. Пароль

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

#### 5. Дialect

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

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

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

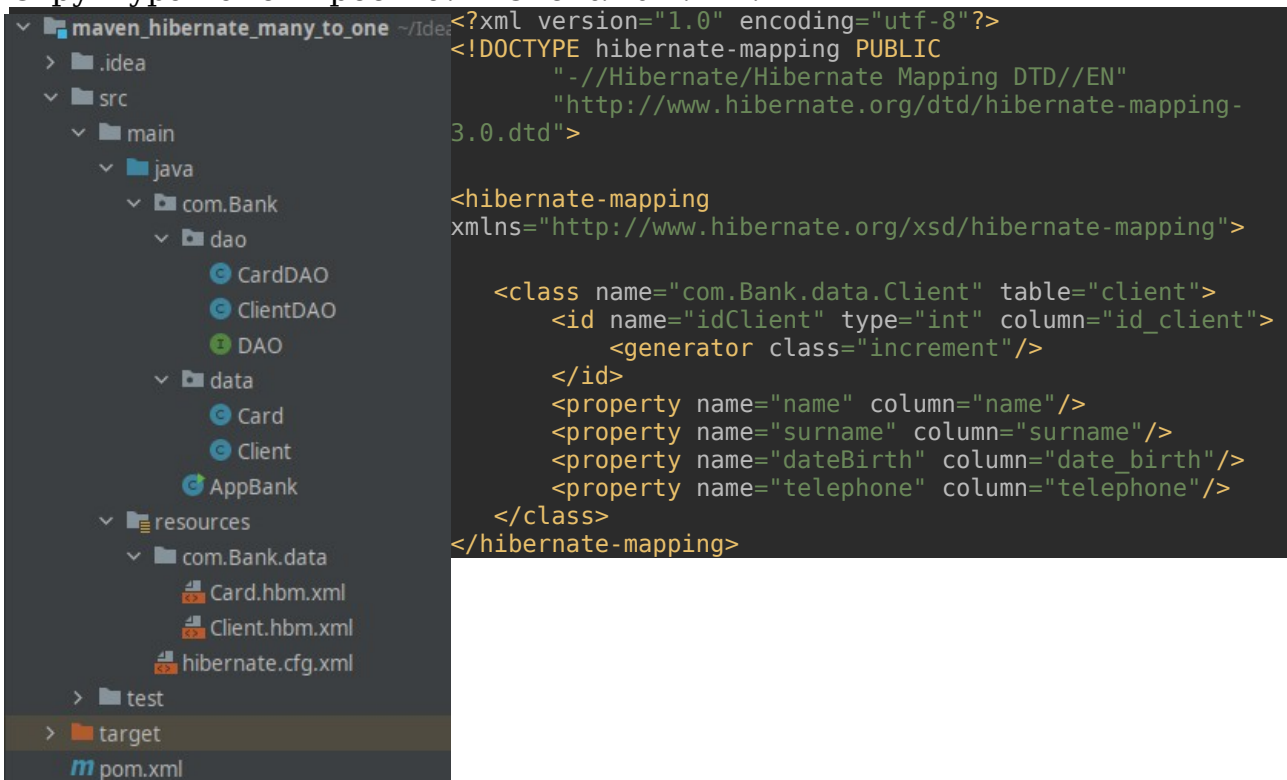
Тегом `mapping`, мы ссылаемся на файл который будет описывать то как соотносятся наши поля в таблице с нашим классом. Для каждого класса, который представляет таблицу должен быть создан такой файл.

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

3. Создаём файлы с названием наших таблиц в БД «Client.hbm.xml», «Card.hbm.xml», в директории, которая в точности повторяет путь до нашего класса Client.

Это конфигурационные файлы hibernate, они должны называться точно также как и классы, к которым они соотносятся, должны повторять структуру, которая есть в папке java, до файлов классов.

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" type="int" column="id_client">
            <generator class="increment"/>
        </id>
        <property name="name" column="name"/>
        <property name="surname" column="surname"/>
        <property name="dateBirth" column="date_birth"/>
        <property name="telephone" column="telephone"/>
    </class>
</hibernate-mapping>
```

Card.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.Card" table="bank_card">
        <id name="idCard" column="id_card">
            <generator class="increment"/>
        </id>
        <property name="type" column="type"/>
        <property name="dateIssue" column="date_issue"/>

        <many-to-one name="client" column="id_client"
            class="com.Bank.data.Client"
            cascade="save-update" />

    </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 property включает все остальные столбцы, представление такое же.

Тег generator с классом increment. Находит максимальное значение столбца и увеличивает на единицу следующее id записи

Маппинг объекта Card к объекту Client:

```
<many-to-one name="client" column="id_client"
    class="com.Bank.data.Client"
    cascade="save-update" />
```

В теге описываем как именно они будут ссылаться, в классе Card это поле будет называться client, а столбец в таблице Client id\_client. Указываем каким классом мы это представляем. Атрибут cascade определяет политику по отношению к дочерним объектам при изменении родительских. При сохранении и обновлении данных каскадно будет изменяться и таблица Client.

4. Создаём класс Client, для хранения 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;
}
}

```

Создём класс Card, для хранения информации о банковской карте:

```

package com.Bank.data;

import java.util.Date;

public class Card {
    private int idCard;
    private String type;
    private Date dateIssue;
    private Client client;
    @Override
    public String toString() {
        return "Card{" +

```

```

        "id_card=" + idCard +
        ", type='" + type + '\'' +
        ", date_issue=" + dateIssue +
        ", client=" + client +
        '}}';
    }
    public int getIdCard() {
        return idCard;
    }
    public String getType() {
        return type;
    }
    public Date getDateIssue() {
        return dateIssue;
    }
    public Client getClient() {
        return client;
    }
    public void setIdCard(int idCard) {
        this.idCard = idCard;
    }
    public void setType(String type) {
        this.type = type;
    }
    public void setDateIssue(Date dateIssue) {
        this.dateIssue = dateIssue;
    }
    public void setClient(Client client) {
        this.client = client;
    }
}

```

5. Создаём интерфейс «DAO», «ClientDAO», «CardDAO» для работы с классами «Client» и «Card».

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;

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

```
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

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

3. сохраняем объект

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

4. Получаем транзакцию и делаем коммит

```
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();
```

CardDAO:

```
package com.Bank.dao;
import com.Bank.data.Card;
import org.hibernate.Hibernate;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

public class CardDAO implements DAO<Card, Integer> {
    private SessionFactory factory;
    public CardDAO(SessionFactory factory) {
        this.factory = factory;
    }
    @Override
    public void create(Card card) {
        try(Session session = factory.openSession()) {
            session.beginTransaction();
            session.save(card);
            session.getTransaction().commit();
        }
    }
    @Override
    public Card read(Integer id) {
        try(Session session = factory.openSession()) {
            Card result = session.get(Card.class, id);
            if(result != null) {
                Hibernate.initialize(result.getClient());
            }
            return result;
        }
    }
    @Override
    public void update(Card card) {
        try(Session session = factory.openSession()) {
            session.beginTransaction();
            session.update(card);
            session.getTransaction().commit();
        }
    }
    @Override
    public void delete(Card card) {
        try(Session session = factory.openSession()) {
            session.beginTransaction();
            session.delete(card);
            session.getTransaction().commit();
        }
    }
}
```



```
    }  
}  
}
```

Здесь следует выделить следующее:

```
    if(result != null) {  
        Hibernate.initialize(result.getClient());  
    }
```

Hibernate.initialize(result.getClient()); когда мы достаём данные hibernate по умолчанию он не достаёт все данные каскадно. Следовательно, когда мы получим объект Card, он не будет содержать данные о сущности из таблицы Client. Для того, чтобы инициализировать все поля объекта, которого мы вернём, применяется метод initialize, куда мы посылаём наш базовый объект и поле, которое мы хотим инициализировать.

6. В классе AppBank прописываем следующий код:

```
package com.Bank;  
  
import com.Bank.dao.CardDAO;  
import com.Bank.data.Card;  
  
import com.Bank.data.Client;  
import com.Bank.dao.ClientDAO;  
  
import com.Bank.dao.DAO;  
  
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();  
  
            /**Класс clientDAO**/  
  
            DAO<Client, Integer> clientDAO = new ClientDAO(factory);  
  
            /**Класс cardDAO**/  
  
            DAO<Card, Integer> cardDao = new CardDAO(factory);  
  
            /** чтение данных из таблицы bank_card **/  
  
            Card result = cardDao.read(1);
```

```

        System.out.println("Read : " + result);
    } 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 следующий код и в консоле отслеживаем результат:

```

Hibernate: select c1_0.id_card,c1_0.id_client,c1_0.date_issue,c1_0.type from bank_card c1_0 where c1_0.id_card=?
Hibernate: select c1_0.id_client,c1_0.date_birth,c1_0.name,c1_0.surname,c1_0.telephone from client c1_0 where c1_0.id_client=?
Read : Card{id_card=1, type='social', date_issue=2022-02-01 00:00:00.0,
client=Client{idClient=1, name='Igor', surname='Verlan', dateBirth=1997-01-23, telephone=68618833}}

```

Данные из таблицы «bank\_card» и «client» прочтены и выведены на экран.  
 Вставляем следующий код в место предыдущего, после комментария и до finally:

```

/** добавление данных в таблицу bank_card и client**/

Card newCard = new Card();
newCard.setType("diamond");
newCard.setDateIssue(Date.valueOf("2022-01-23"));
Client client = new Client();
client.setName("Aurel");
client.setSurname("Gonga");
client.setDateBirth(Date.valueOf("1987-11-01"));
client.setTelephone(60045103);
newCard.setClient(client);
System.out.println(newCard);
cardDao.create(newCard);

```

Запускаем. Проверяем работу программы:

```

Card{id_card=0, type='diamond', date_issue=2022-01-23,
client=Client{idClient=0, name='Aurel', surname='Gonga', dateBirth=1987-11-01, telephone=60045103}}
Hibernate: select max(id_card) from bank_card
Hibernate: select max(id_client) from client
Hibernate: insert into client (date_birth, name, surname, telephone, id_client) values (?, ?, ?, ?, ?)
Hibernate: insert into bank_card (id_client, date_issue, type, id_card) values (?, ?, ?, ?)

```

Проверяем в базе данных:

```
BankDB=# select *
BankDB=# from client
BankDB=# join bank_card
BankDB=# on bank_card.id_client = client.id_client;
 id_client | name  | surname | date_birth | telephone | id_card | type   | date_issue | id_client
-----+-----+-----+-----+-----+-----+-----+-----+-----
          1 | Igor  | Verlan  | 1997-01-23 | 68618833 |        1 | social | 2022-02-01 |          1
          2 | Aurel | Gonga   | 1987-11-01 | 60045103 |        2 | diamond | 2022-01-23 |          2
(2 строки)
```

Новые данные введены в обе таблицы.

Проверяем результат работы следующего кода:

```
/** добавление данных только в таблицу bank_card**/

Card newCard = new Card();
newCard.setType("gold");
newCard.setDateIssue(Date.valueOf("2021-01-23"));
Client client = clientDAO.read(2);
newCard.setClient(client);
cardDao.create(newCard);
```

Результат:

```
Hibernate: select c1_0.id_client,c1_0.date_birth,c1_0.name,c1_0.surname,c1_0.telephone from client c1_0 where c1_0.id_client=?
Hibernate: select max(id_card) from bank_card
Hibernate: insert into bank_card (id_client, date_issue, type, id_card) values (?, ?, ?, ?)
Hibernate: update client set date_birth=?, name=?, surname=?, telephone=? where id_client=?
```

```
BankDB=# select * from bank_card;
 id_card | type   | date_issue | id_client
-----+-----+-----+-----
          1 | social | 2022-02-01 |          1
          2 | diamond | 2022-01-23 |          2
          3 | gold   | 2021-01-23 |          2
(3 строки)
```

Следующий код обновляет данные о карте и о её владельце:

```
/** обновление данных в таблице bank_card и client**/

Card result = cardDao.read(2);
System.out.println("Read : " + result);
result.setType("business");
result.getClient().setTelephone(78005405);
cardDao.update(result);
System.out.println("Update : " + cardDao.read(2));
```

Результат:

```
Hibernate: select cl_0.id_card,cl_0.id_client,cl_0.date_issue,cl_0.type from bank_card cl_0 where cl_0.id_card=?
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 : Card{id_card=2, type='diamond', date_issue=2022-01-23 00:00:00.0,
client=Client{idClient=2, name='Aurel', surname='Gonga', dateBirth=1987-11-01, telephone=60045103}}
Hibernate: update bank_card set id_client=?, date_issue=?, type=? where id_card=?
Hibernate: update client set date_birth=?, name=?, surname=?, telephone=? where id_client=?
Hibernate: select cl_0.id_card,cl_0.id_client,cl_0.date_issue,cl_0.type from bank_card cl_0 where cl_0.id_card=?
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=?
Update : Card{id_card=2, type='business', date_issue=2022-01-23 00:00:00.0,
client=Client{idClient=2, name='Aurel', surname='Gonga', dateBirth=1987-11-01, telephone=78005405}}
```

```
BankDB=# select *
from client
join bank_card
on bank_card.id_client = client.id_client;
 id_client | name  | surname | date_birth | telephone | id_card | type   | date_issue | id_client
-----+-----+-----+-----+-----+-----+-----+-----+-----
          1 | Igor  | Verlan  | 1997-01-23 | 68618833 |         1 | social | 2022-02-01 |          1
          2 | Aurel | Gonga   | 1987-11-01 | 78005405 |         3 | gold   | 2021-01-23 |          2
          2 | Aurel | Gonga   | 1987-11-01 | 78005405 |         2 | business | 2022-01-23 |          2
(3 строки)
```

Изменились телефонный номер клиента под номером два и тип банковской карты под номером два.

Следующий код удаляет банковскую карту:

```
/**удаление данных таблицы bank_card**/

Card result = cardDao.read(2);
System.out.println("Read : " + result);
cardDao.delete(result);
```

Глядим на результат:

```
Hibernate: select cl_0.id_card,cl_0.id_client,cl_0.date_issue,cl_0.type from bank_card cl_0 where cl_0.id_card=?
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 : Card{id_card=2, type='business', date_issue=2022-01-23 00:00:00.0,
client=Client{idClient=2, name='Aurel', surname='Gonga', dateBirth=1987-11-01, telephone=78005405}}
Hibernate: delete from bank_card where id_card=?
```

```
BankDB=# select * from bank_card;
 id_card | type   | date_issue | id_client
-----+-----+-----+-----
          1 | social | 2022-02-01 |          1
          3 | gold   | 2021-01-23 |          2
(2 строки)
```