Connection ( а в Hibernate Session)

Это средство, позволяющее программному обеспечению клиента взаимодействовать с программным обеспечением сервера базы данных,

Для подключения к БД нужен хост, порт(на каком порту слушает сервер БД), пароль и БД (название)

Клиент меняет БД с помощью транзакций

Transaction( транзакция)

Это последовательное выполнение операций чтения и записи.

**UPDATE users SET money = money + 2000 WHERE name = 'Надя’**

**||**

**DELETE FROM users WHERE name = 'Надя’**

(ни апдейт, ни делит не происходят мнгновенно)

Окончанием транзакции может быть либо сохранение изменений (фиксация, commit) либо отмена изменений (откат, rollback)

Т.е. чтобы сделать фиксацию в транзакции, нужно сделать коммит. ОН может быть явным, может быть неявным)

Commit

Когда транзакция завершается, нужно сделать commit в БД по аналогии как мы делаем commit в Git чтобы внести изменения совершенные за транзакцию в БД.

**INSERT INTO table VAlUES(….) COMMIT;**

В некоторых БД коммит стоит автоматически

Зачем БД в автотестах:

**Получение тестовых данных**

**Проверки изменения данных в БД**

**Изменение данных в БД**

**JDBC** (это библиотека) (англ. Java DataBase Connectivity — соединение с базами данных на Java) — платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

Она уже все умеет, может селектить, апдейтить и т.п.

**JDBC-драйвер** – реализация JDBC для определенной базы данных.

В приложении может быть зарегистрировано несколько разных драйверов.

Объявление драйвера в классе

Class.*forName*("org.postgresql.Driver");

Класс **DriverManager** является уровнем управления JDBC, отслеживает все доступные драйверы и управляет установлением соединений между БД и соответствующим драйвером.

Установка соединения: DriverManager.*getConnection*(*DB\_URL*,*USER*,*PASS*)

Класс **java.sql.Connection** представляет в JDBC сеанс работы с базой данных. Он предоставляет приложению объекты Statement (и его подклассы) для этого сеанса.

Создание объекта Statement:

Statement statement = connection.createStatement();

Класс Statement представляет встроенную команду SQL и используется приложением для доступа к базе данных.

Это сущность, которая может общаться с БД

(update, execute, insert)

ResultSet executeQuery(String sql) throws SQLException;

Коннекшн надо закрывать после окончания работы с БД, потому что количество коннекшинов ограничено

Ну представьте, что у Вас 3000 автотестов и каждый хочет свой коннекшн

Если закрываете коннекшин, то стейтмент закрывается автоматически

если хотим только закрыть БД, но оставить коннекшн, то закрыть Стейтмент и оставить коннекшн

**ResultSet executeQuery(String sql) throws SQLException;**

**int executeUpdate(String sql) throws SQLException;**

**void close() throws SQLException;**

**executeUpdate** возвращает количество строк поэтому int

int executeUpdate (String sql)

**void close ()** нужен так как у БД рано или поздно может закончится количество коннекшинов

**ResultSet**

Это класс. Это поток к результату выполнения вашего селект запроса

Нужен для построчного доступа. Если сразу все строки из БД вытащить, то приложение может упасть, а с резалт сет этого можно избежать

ResultSet создает поток данных, пот которому она может выдергивать по одному выдергивать строку за строкой

Получается что-то вроде итератора

boolean next() throws SQLException

(проверяет, что есть следующая строка в ответе)

boolean previous() throws SQLException;

(проверяет, что есть предыдущая строка в ответе)

String getString(int columnIndex) throws SQLException;

Драйвер для каждой БД надо прописывать вручную. Он не возьмется из ниоткуда

И прописать в градле

implementation group: 'postgresql', name: 'postgresql', version: '9.1-901-1.jdbc4'

ЗАВИСИМОСТЬ В ГРАДДЛЕ НАДО ДОБАВИТЬ

**Ctrl +alt+ t чтобы обернуть в try catch**

Настройка GetConnection для JDBC

public class DBClient {  
 private Connection connection = null;  
 private Connection getConnection() {  
 if(connection == null){  
 try {  
 try {  
 Class.*forName*("org.postgresql.Driver");  
 connection = DriverManager.*getConnection*("jdbc:postgresql:localhost:5437/postgres","fintech","password");  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return connection;  
 }  
  
}

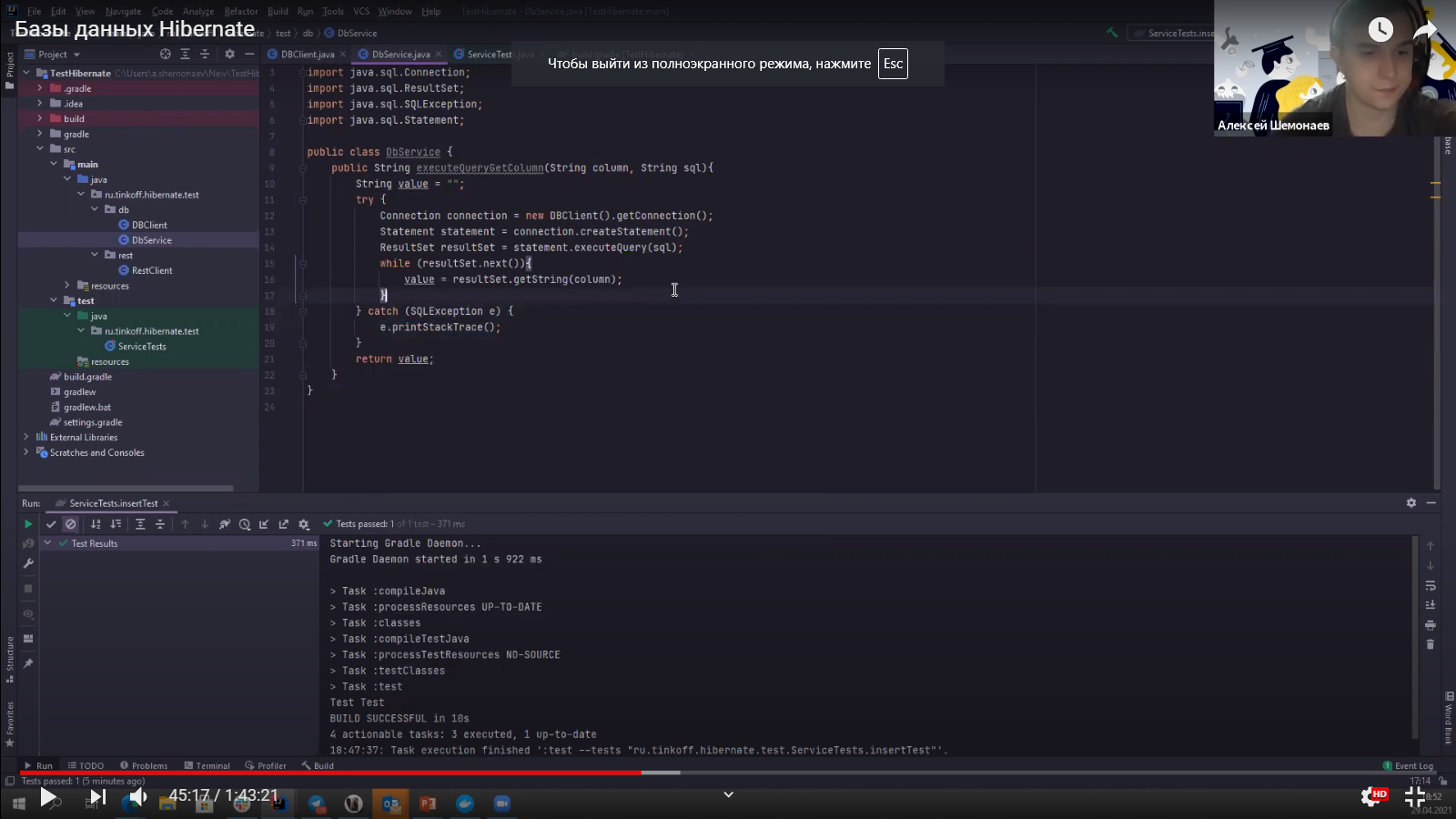
Session для Hibernate

import org.hibernate.cfg.Configuration;  
  
  
public class HibernateSessionCreator {  
  
 private static SessionFactory *sessionFactory* = null;  
  
 public static SessionFactory getSessionFactory(){  
 if (*sessionFactory* == null) {  
 *sessionFactory* = *buildSessionFactory*();  
 }  
 return *sessionFactory*;  
 }  
  
 private static SessionFactory buildSessionFactory(){  
 return new Configuration()  
 .configure()  
 .addAnnotatedClass(User.class)  
 .buildSessionFactory();  
  
 }  
}

Когда экземпляр класса Configuration разберет (parse) все отображения/мэппинги, приложение получает фабрику (factory) для экземпляров класса Session. Эта фабрика должна быть распределена (shared) между всеми потоками приложения:

Configuration xml:

<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.show\_sql">true</property>  
 <property name="hibernate.connection.driver\_class">org.postgresql.Driver</property>  
 <property name="hibernate.connection.username">fintech</property>  
 <property name="hibernate.connection.password">password</property>  
 <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/postgres</property>  
 <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQL9Dialect</property>  
 </session-factory>  
</hibernate-configuration>



package ru.fintech.qa.homework.utils.db;  
  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class DBService {  
 public String executeQueryGetColumn (String column, String sql){  
 String value = "";  
 Connection connection = new DBClient().getConnection();  
 try {  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery(sql);  
 while (resultSet.next()){  
 value = resultSet.getString(column);  
 }  
 connection.close();  
 } catch (SQLException e) {  
 try {  
 connection.close();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 e.printStackTrace();  
 }  
 return value;  
 }  
}

Hibernate

Hibernate — это решение для JAVA. Это постоянный фреймворк с открытым исходным кодом.

Это ORM

Hibernate занимается отображением классов Java в таблицы базы данных с использованием файлов XML и без написания какой-либо строки кода.

Предоставляет простые API-интерфейсы для работы с БД

Абстрагирует незнакомые типы SQL в типы Java

Зависимости

**implementation 'org.hibernate:hibernate-core:5.2.6.Final'  
  
implementation group: 'postgresql', name: 'postgresql', version: '9.1-901-1.jdbc4'**

**Configuration**

**(добавляется в качестве xml файла в проект в resources)**

Это сущность, позволяющая настроить соединение с базой данных. Может быть представлено в виде .xml .yaml .properties или настраивается вручную.

**В БД можно хранить JSON файлы**

*<?*xml version="1.0" encoding="utf-8"*?>*<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd"*>*<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.show\_sql">true</property>  
 <property name="hibernate.connection.driver\_class">org.postgresql.Driver</property>  
 <property name="hibernate.connection.username">fintech</property>  
 <property name="hibernate.connection.password">password</property>  
 <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/postgres</property>  
 <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQL9Dialect</property>

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

**Диалекты**

Oracle (любая версия)

org.hibernate.dialect.OracleDialect

MySQL

org.hibernate.dialect.MySQLDialect

PostgreSQL

org.hibernate.dialect.PostgreSQLDialect

HSQLDB

org.hibernate.dialect.HSQLDialect

Hibernate работает с БД как с моделью данных

И каждая таблица для Хибернейт – класс

Под **моделью данных** в hibernate понимается класс полностью соответствующий таблице в базе данных. То есть у класса все поля полностью повторяют порядок, тип и название столбцов в таблице.

Аннотации:

@Entity  
@Table(name = "user")  
public class User {  
 @Id  
 int id;  
 @Column(name = "fio")  
 String fio;  
 @Column(name = "phone")  
 String phone;  
 <getters and setters>  
}

@Entity Аннотация нужна для пометки класса как сущность для Hibernate. А также сообщает, что является не только сущностью в Хибернейте, а также и таблицей в БД

@Table(name = "user") Помечает класс как таблицу, и уточняет к какой именно таблице относится класс

@Id

Указывает на primary key в таблице

@Column(name = "fio") Указывает какое поле в классе относится к какому столбцу в базе данных.

**@OneToMany \ @ManyToOne**

@Entity  
@Table(name = "user")  
@Data  
public class User {  
 @Id  
 int id;  
 @Column(name = "fio")  
 String fio;  
 @Column(name = "phone")  
 String phone;  
 @OneToMany  
 List<Pets> pets;  
}

@Entity  
@Table(name = "pets")  
@Data  
class Pets {  
 @Id  
 int id;  
 @Column(name = "name\_pet")  
 String name;  
 @Column(name = "user")  
 @ManyToOne  
 User user;  
}

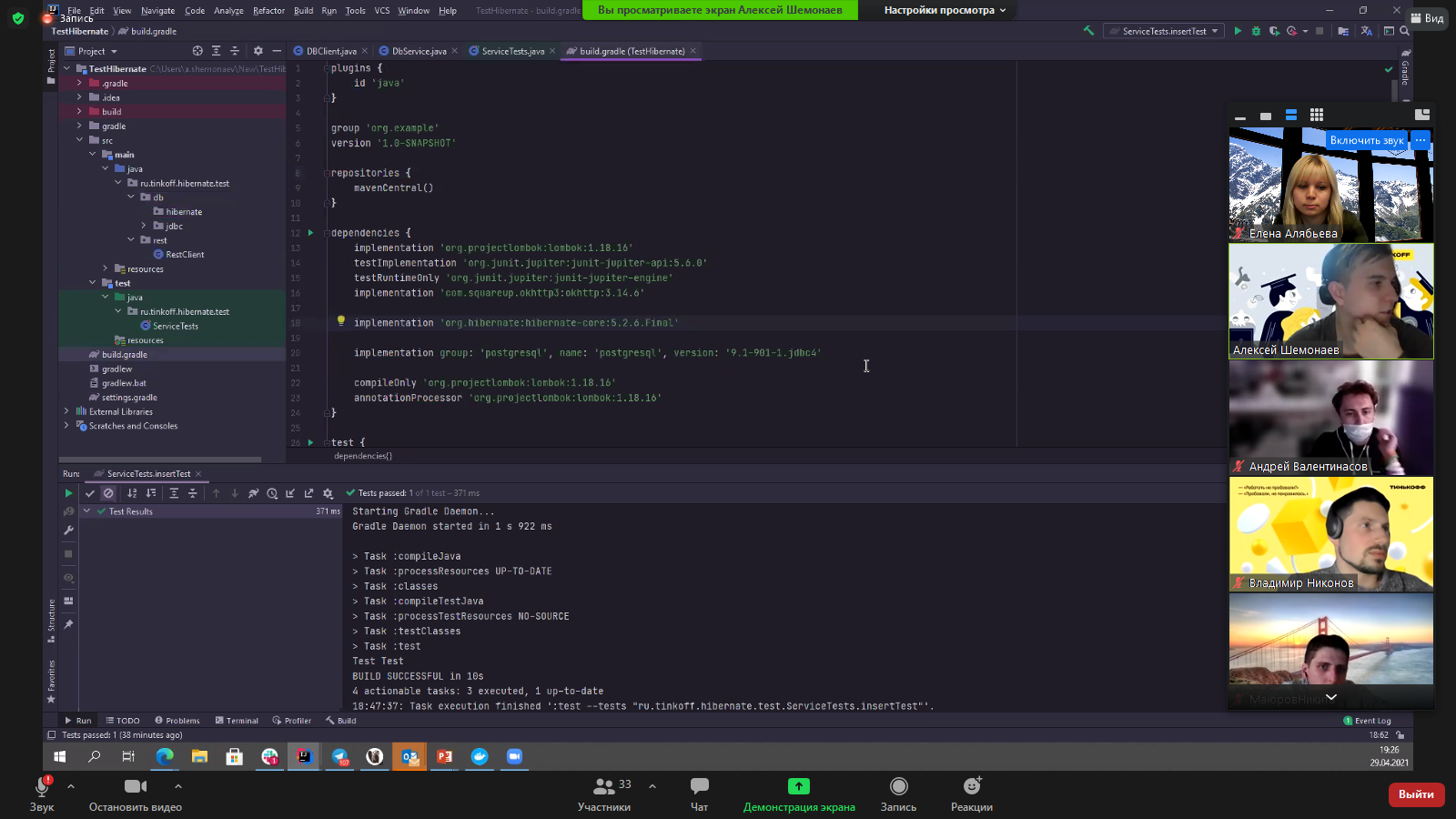
**Session**

Сессия используется для получения физического соединения с базой данных.

Объект Session является легким и предназначен для реализации каждый раз,

когда необходимо взаимодействие с базой данных.

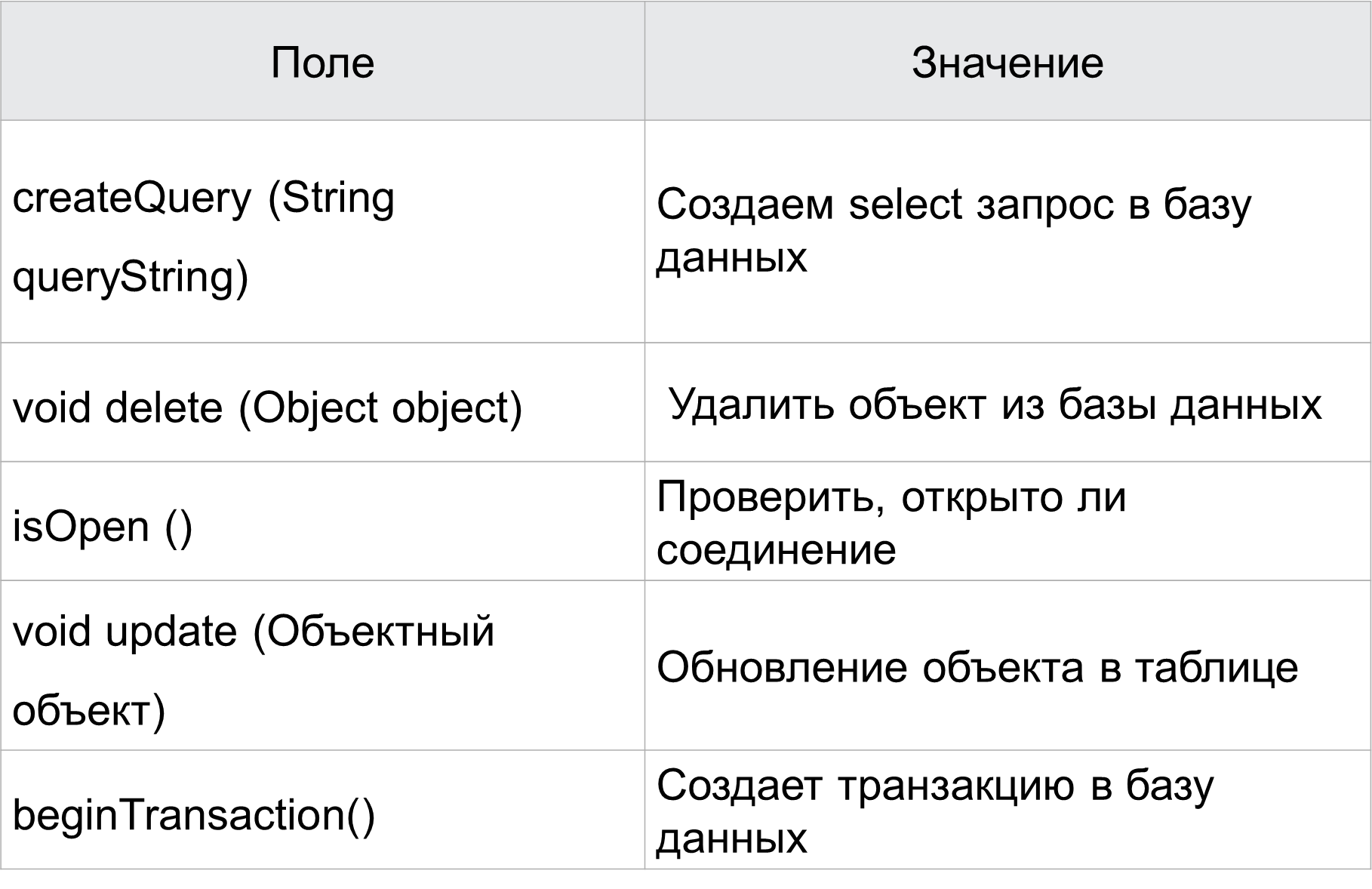
Постоянные объекты сохраняются и извлекаются через объект Session.



private static SessionFactory buildNewSessionFactory() {  
 return new Configuration()  
 .configure()  
 .addAnnotatedClass(DataItem.class)  
 .buildSessionFactory();  
}

.configure() – означает что хибернейт по дефолту будет искать xml с настройками. Yaml тоже можно, но это надо явно указывать (формат данных, где лежит файл и т.п.)

.addAnnotatedClass(DataItem.class) –явно указываем хибернейту, на какие файлы нужно смотреть



ALT+INSERT помогает быстро сформировать все геттеры и сеттеры

Так как начиная с 9 версии Java в ней больше нет JAX, то чтобы не выдавалась ошибка, нужно в градле в зависимостях прописать

testCompile group: 'org.glassfish.jaxb', name: 'jaxb-runtime', version: '2.3.1'

или так:

implementation "jakarta.xml.bind:jakarta.xml.bind-api:2.3.2"  
implementation "org.glassfish.jaxb:jaxb-runtime:2.3.2"