Министерство транспорта Российской Федерации

Федеральное государственное автономное образовательное  
учреждение высшего образования  
«РУТ (МИИТ)»

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

КУРСОВОЙ ПРОЕКТ

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему  
«Магазин гель-лаков»**

Выполнил: ст. гр. ТКИ-541

Станчук П. Н.

Проверил: доц., к.т.н.

Васильева М. А.

Москва 2023

**Оглавление**

[***ЦЕЛЬ КУРСОВОГО ПРОЕКТА 4***](#_1fob9te)

[***ЗАДАНИЕ НА КУРСОВОВОЙ ПРОЕКТ 5***](#_3znysh7)

[***ПРОЕКТИРОВАНИЕ URM 6***](#_tyjcwt)

[**1. Инфологическое проектирование 6**](#_3dy6vkm)

[1.1 Анализ предметной области 6](#_1t3h5sf)

[1.2 Анализ информационных задач и круга пользователей системы 7](#_2s8eyo1)

[**2. Логическое проектирование реляционной БД 7**](#_17dp8vu)

[2.1 Составление реляционных отношений 7](#_3rdcrjn)

[**3. Код программы 11**](#_2jxsxqh)

[**4. Тесты 27**](#_z337ya)

[***Список литературы 43***](#_3j2qqm3)

# ЦЕЛЬ КУРСОВОГО ПРОЕКТА

Целью курсового проекта является изучение методов и закрепление знаний в создании URM для реляционных баз данных (РБД) в системе управления базами данных (СУБД) Система управления базами данных PostgreSQL.

# ЗАДАНИЕ НА КУРСОВОВОЙ ПРОЕКТ

В данном курсовом проекте ставится задача разработать URM для СУБД PostgreSQL по заданной теме «Магазин гель-лаков». Проектирование РБД проводится с помощью метода «Сущность-связь». Проверка построенной модели РБД осуществляется с помощью метода нормализации отношений.

**Описание предметной области.**

БД создается для учёта продажи гель-лаков в магазинах от разных поставщиков.

**Готовые запросы:**

• Выдавать список клиентов, купивших товар в конкретном магазине.

• Выдавать список поставщиков, поставляющих товар в данный магазин.

• Выдавать список клиентов из данной страны.

• Выдавать количество поставщиков для данного склада.

# ПРОЕКТИРОВАНИЕ URM

## Инфологическое проектирование

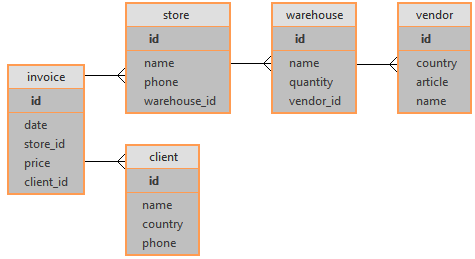
### Анализ предметной области

База данных создаётся для информационного обслуживания продаж лаков в магазинах.

Выделим базовые сущности этой предметной области:

* Клиент
* Поставщик
* Склад
* Накладная
* Магазин

ER-диаграмма БД, приведена на (Рисунке 1). На данной ER-диаграмме отсутствует связь многие -ко– многим, поэтому дополнительная промежуточная сущностей не требуется.



**Рисунок 1 – ER-диаграмма**

### Анализ информационных задач и круга пользователей системы

Определим границы информационной поддержки пользователей:

**1) Функциональные возможности:**

* ведение БД (запись, чтение, модификация, удаление);
* обеспечение логической непротиворечивости БД;
* реализация наиболее часто встречающихся запросов в готовом виде.

**2) Готовые запросы:**

* Выдавать список клиентов, купивших товар в конкретном магазине.
* Выдавать список поставщиков, поставляющих товар в данный магазин.
* Выдавать список клиентов из данной страны.
* Выдавать количество поставщиков для данного склада.

## Логическое проектирование реляционной БД

### Составление реляционных отношений

Каждое реляционное отношение соответствует одной сущности (объекту ПрО), и в него вносятся все атрибуты сущности. Для каждого отношения необходимо определить первичный ключ и внешние ключи (если они есть). В том случае, если базовое отношение не имеет потенциальных ключей, вводится *суррогатный первичный ключ*, который не несёт смысловой нагрузки и служит только для идентификации записей.

Были созданы следующие сущности:

1. Таблица "store" - содержит информацию о магазине.
2. Таблица "warehouse" – содержит информацию о складе.
3. Таблица "client" - содержит список клиентов.
4. Таблица "vendor" - содержит список поставщиков.
5. Таблица "invoice" - содержит список накладных.

**Таблица 1 – Схема отношения** **Магазин** (store)

| **Содержание поля** | **Имя поля** | **Тип** | **Длина** | **Примечания** |
| --- | --- | --- | --- | --- |
| Идентификатор | id | Целое число | - | Первичный ключ, автоинкремент |
| Имя | name | Текст | - | Обязательное поле |
| Телефон | phone | Текст | - | Обязательное поле |
| Идентификатор склада | warehouse\_id | Целое число | - | Внешний ключ |

**Таблица 2 – Схема отношения Поставщик** (Vendor)

| **Содержание поля** | **Имя поля** | **Тип** | **Длина** | **Примечания** |
| --- | --- | --- | --- | --- |
| Идентификатор | id | Целое число | - | Первичный ключ, автоинкремент |
| Страна | country | Текст | - | Обязательное поле |
| Артикул | article | Целое число | - | Обязательное поле |
| Название | name | Текст | - | Обязательное поле |

**Таблица 3 – Схема отношения Склад** (Warehouse)

| **Содержание поля** | **Имя поля** | **Тип** | **Длина** | **Примечания** |
| --- | --- | --- | --- | --- |
| Идентификатор | id | Целое число | - | Первичный ключ, автоинкремент |
| Название | name | Текст | - | Обязательное поле |
| Количество товаров | quantity | Целое число | - | Обязательное поле |
| Идентификатор поставщика | id | Целое число | - | Внешний ключ |

**Таблица 4 – Схема отношения Клиент** (Client)

| **Содержание поля** | **Имя поля** | **Тип** | **Длина** | **Примечания** |
| --- | --- | --- | --- | --- |
| Идентификатор | id | Целое число | - | Первичный ключ, автоинкремент |
| ФИО | name | Текст | - | Обязательное поле |
| Страна | country | Текст | - | Обязательное поле |
| Телефон | phone | Текст | - | Обязательное поле |

**Таблица 5 – Схема отношения Накладная** (Invoice)

| **Содержание поля** | **Имя поля** | **Тип** | **Длина** | **Примечания** |
| --- | --- | --- | --- | --- |
| Идентификатор | id | Целое число | - | Первичный ключ, автоинкремент |
| Дата продажи | date | Дата | - | Обязательно поле |
| Цена | price | Денежный | - | Обязательное поле |
| Идентификатор магазина | store\_id | Целое число | - | Внешний ключ |
| Идентификатор клиента | client\_id | Целое число | - | Внешний ключ |

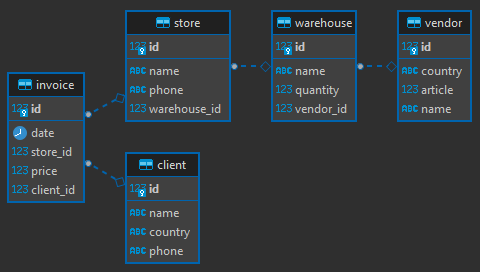
Нормализация полученных отношений

**1НФ.** Для приведения таблиц к 1НФ требуется составить таблицы (один атрибут – один столбец) и разбить сложные атрибуты на простые, а многозначные атрибуты вынести в отдельные отношения.  
В наших таблицах все атрибуты простые.

**2НФ.** Все отношения находятся во 2НФ, так как для каждого из них введен уникальный ключ.

**3НФ**. Все неключевые атрибуты зависят только от первичного ключа и не зависят друг от друга. Поэтому спроектированная **база данных находится в третьей нормальной форме**.

Созданную схему Вы можете наблюдать на рисунке 2.



**Рисунок 2 – ER-диаграмма в DBeaver**

## Код программы

| **Файл BaseMethod.java** package org.example;  import org.hibernate.Session; import org.hibernate.query.Query; import javax.persistence.criteria.CriteriaBuilder; import javax.persistence.criteria.CriteriaQuery; import javax.persistence.criteria.Root; import java.util.List;  public class BaseMethod<T> {   private final Session session;  private final Class<T> entityClass;   public BaseMethod(Session session, Class<T> entityClass) {  this.session = session;  this.entityClass = entityClass;  }   public T get(int id) {  return session.get(entityClass, id);  }   public List<T> find(String field, Object value) {  CriteriaBuilder builder = session.getCriteriaBuilder();  CriteriaQuery<T> criteria = builder.createQuery(entityClass);  Root<T> root = criteria.from(entityClass);  criteria.select(root).where(builder.equal(root.get(field), value));  Query<T> query = session.createQuery(criteria);  return query.getResultList();  }   public List<T> getAll() {  CriteriaBuilder builder = session.getCriteriaBuilder();  CriteriaQuery<T> criteria = builder.createQuery(entityClass);  criteria.from(entityClass);  Query<T> query = session.createQuery(criteria);  return query.getResultList();  }   public void save(T entity) {  session.beginTransaction();  session.saveOrUpdate(entity);  session.getTransaction().commit();  }   public void delete(T entity) {  session.beginTransaction();  session.delete(entity);  session.getTransaction().commit();  } }  **Файл Client.java** package org.example;  import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany; import java.util.List;  @Entity @Table(name = "client") public class Client {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private int id;   @Column(name = "name")  private String name;   @Column(name = "country")  private String country;   @Column(name = "phone")  private String phone;   @OneToMany(mappedBy = "client")  private List<Invoice> invoices;   // Пустой конструктор для Hibernate  public Client() {  }   public int getId() {  return id;  }   private void setId(int id) {  this.id = id;  }   public String getName() {  return name;  }   private void setName(String name) {  if (name == null || name.isEmpty()) {  throw new IllegalArgumentException("Имя клиента не может быть пустым.");  }  this.name = name;  }   public String getCountry() {  return country;  }   private void setCountry(String country) {  if (country == null || country.isEmpty()) {  throw new IllegalArgumentException("Страна клиента не может быть пустой.");  }  this.country = country;  }   public String getPhone() {  return phone;  }   private void setPhone(String phone) {  if (phone == null || phone.isEmpty()) {  throw new IllegalArgumentException("Телефон клиента не может быть пустым.");  }  this.phone = phone;  }   public List<Invoice> getInvoices() {  return invoices;  }   private void setInvoices(List<Invoice> invoices) {  this.invoices = invoices;  } }  **Файл Invoice.java** package org.example;  import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany; import java.util.List;  @Entity @Table(name = "invoice") public class Invoice {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private int id;   @Column(name = "date")  private Date date;   @Column(name = "price")  private double price;   @ManyToOne  @JoinColumn(name = "store\_id")  private Store store;   @ManyToOne  @JoinColumn(name = "client\_id")  private Client client;   // Пустой конструктор для Hibernate  public Invoice() {  }   public int getId() {  return id;  }   private void setId(int id) {  this.id = id;  }   public Date getDate() {  return date;  }   private void setDate(Date date) {  if (date == null) {  throw new IllegalArgumentException("Дата счета не может быть пустой.");  }  this.date = date;  }   public double getPrice() {  return price;  }   private void setPrice(double price) {  if (price <= 0) {  throw new IllegalArgumentException("Сумма счета должна быть больше нуля.");  }  this.price = price;  }   public Store getStore() {  return store;  }   private void setStore(Store store) {  if (store == null) {  throw new IllegalArgumentException("Магазин счета не может быть пустым.");  }  this.store = store;  }   public Client getClient() {  return client;  }   private void setClient(Client client) {  if (client == null) {  throw new IllegalArgumentException("Клиент счета не может быть пустым.");  }  this.client = client;  } }  **Файл Main.java**  package org.example;  import org.hibernate.Session; import org.hibernate.SessionFactory; import org.hibernate.cfg.Configuration;  import java.util.Date; import java.util.List; import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany; import java.util.List;  public class Main {   public static void main(String[] args) {  // Создаем конфигурацию Hibernate  Configuration configuration = new Configuration().configure("hibernate.cfg.xml");   // Создаем фабрику сессий  SessionFactory sessionFactory = configuration.buildSessionFactory();   // Создаем сессию для взаимодействия с базой данных  Session session = sessionFactory.openSession();   try {  // Создание экземпляров BaseMethod для каждой сущности  BaseMethod<Store> storeMethod = new BaseMethod<>(session, Store.class);  BaseMethod<Warehouse> warehouseMethod = new BaseMethod<>(session, Warehouse.class);  BaseMethod<Client> clientMethod = new BaseMethod<>(session, Client.class);  BaseMethod<Vendor> vendorMethod = new BaseMethod<>(session, Vendor.class);  BaseMethod<Invoice> invoiceMethod = new BaseMethod<>(session, Invoice.class);   // Создание и сохранение объектов   // Пример создания и сохранения склада (Warehouse)  Warehouse warehouse = new Warehouse("Название склада", 100);  warehouseMethod.save(warehouse);   // Пример создания и сохранения магазина (Store)  Store store = new Store("Название магазина", "123-456-7890", warehouse);  storeMethod.save(store);   // Пример создания и сохранения клиента (Client)  Client client = new Client("Имя клиента", "Страна клиента", "987-654-3210");  clientMethod.save(client);   // Пример создания и сохранения поставщика (Vendor)  Vendor vendor = new Vendor("Страна поставщика", 123456, "Название поставщика");  vendorMethod.save(vendor);   // Пример создания и сохранения счета (Invoice)  Invoice invoice = new Invoice(new Date(), 100.0, store, client);  invoiceMethod.save(invoice);   // Пример установки связей между объектами  store.setWarehouse(warehouse);  invoice.setStore(store);  invoice.setClient(client);  warehouse.setVendor(vendor);   // Получение и вывод всех объектов определенной сущности  List<Store> allStores = storeMethod.getAll();  for (Store s : allStores) {  System.out.println("Название магазина: " + s.getName());  System.out.println("Телефон: " + s.getPhone());  }   // По аналогии можно получать и выводить другие сущности   } finally {  // Закрываем сессию и фабрику сессий  session.close();  sessionFactory.close();  }  } }  **Файл Store.java** package org.example;  import javax.persistence.\*; import java.util.List; import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany;  @Entity @Table(name = "store") public class Store {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private int id;   @Column(name = "name")  private String name;   @Column(name = "phone")  private String phone;   @ManyToOne  @JoinColumn(name = "warehouse\_id")  private Warehouse warehouse;   @OneToMany(mappedBy = "store")  private List<Invoice> invoices;   // Пустой конструктор для Hibernate  public Store() {  }   public int getId() {  return id;  }   private void setId(int id) {  this.id = id;  }   public String getName() {  return name;  }   private void setName(String name) {  if (name == null || name.isEmpty()) {  throw new IllegalArgumentException("Название магазина не может быть пустым.");  }  this.name = name;  }   public String getPhone() {  return phone;  }   private void setPhone(String phone) {  if (phone == null || phone.isEmpty()) {  throw new IllegalArgumentException("Телефон магазина не может быть пустым.");  }  this.phone = phone;  }   public Warehouse getWarehouse() {  return warehouse;  }   private void setWarehouse(Warehouse warehouse) {  this.warehouse = warehouse;  }   public List<Invoice> getInvoices() {  return invoices;  }   private void setInvoices(List<Invoice> invoices) {  this.invoices = invoices;  } }  **Файл Vendor.java** package org.example;  import javax.persistence.\*; import java.util.List; import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany;  @Entity @Table(name = "vendor") public class Vendor {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private int id;   @Column(name = "country")  private String country;   @Column(name = "article")  private int article;   @Column(name = "name")  private String name;   @OneToMany(mappedBy = "vendor")  private List<Warehouse> warehouses;   // Пустой конструктор для Hibernate  public Vendor() {  }   public Vendor(String country, int article, String name) {  setCountry(country);  setArticle(article);  setName(name);  }   public int getId() {  return id;  }   private void setId(int id) {  this.id = id;  }   public String getCountry() {  return country;  }   private void setCountry(String country) {  if (country == null || country.isEmpty()) {  throw new IllegalArgumentException("Страна поставщика не может быть пустой.");  }  this.country = country;  }   public int getArticle() {  return article;  }   private void setArticle(int article) {  if (article <= 0) {  throw new IllegalArgumentException("Артикул поставщика должен быть больше нуля.");  }  this.article = article;  }   public String getName() {  return name;  }   private void setName(String name) {  if (name == null || name.isEmpty()) {  throw new IllegalArgumentException("Название поставщика не может быть пустым.");  }  this.name = name;  }   public List<Warehouse> getWarehouses() {  return warehouses;  }   private void setWarehouses(List<Warehouse> warehouses) {  this.warehouses = warehouses;  } }  **Файл Warehouse.java** package org.example;  import javax.persistence.\*; import java.util.Date; import javax.persistence.Entity; import javax.persistence.Table; import javax.persistence.Id; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Column; import javax.persistence.OneToMany; import java.util.List;  @Entity @Table(name = "warehouse") public class Warehouse {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id")  private int id;   @Column(name = "name")  private String name;   @Column(name = "quantity")  private int quantity;   @ManyToOne  @JoinColumn(name = "vendor\_id")  private Vendor vendor;   // Пустой конструктор для Hibernate (обычно требуется Hibernate)  public Warehouse() {  }   // Геттер и сеттер для поля "id"  public int getId() {  return id;  }   // Приватный сеттер для поля "id", чтобы его нельзя было изменить извне  private void setId(int id) {  this.id = id;  }   // Геттер и сеттер для поля "name" с проверкой на пустое значение  public String getName() {  return name;  }   public void setName(String name) {  if (name == null || name.isEmpty()) {  throw new IllegalArgumentException("Название склада не может быть пустым.");  }  this.name = name;  }   // Геттер и сеттер для поля "quantity" с проверкой на отрицательное значение  public int getQuantity() {  return quantity;  }   public void setQuantity(int quantity) {  if (quantity < 0) {  throw new IllegalArgumentException("Количество товара на складе не может быть отрицательным.");  }  this.quantity = quantity;  }   // Геттер и сеттер для поля "vendor"  public Vendor getVendor() {  return vendor;  }   public void setVendor(Vendor vendor) {  this.vendor = vendor;  } } |
| --- |

## Тесты

| package test;  mport org.junit.jupiter.api.BeforeEach;  import org.junit.jupiter.api.Test; import static org.junit.jupiter.api.Assertions.\*; import java.util.Arrays; import java.util.List;  import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import static org.junit.jupiter.api.Assertions.\*; import org.example.BaseMethod; import org.example.Client;  public class BaseMethodTest {   private static SessionFactory sessionFactory;  private static Session session;  private static BaseMethod<Client> clientBaseMethod;   @BeforeAll  static void setUpBeforeClass() {  Configuration configuration = new Configuration().configure("hibernate.cfg.xml");  sessionFactory = configuration.buildSessionFactory();  session = sessionFactory.openSession();  clientBaseMethod = new BaseMethod<>(session, Client.class);  }   @AfterAll  static void tearDownAfterClass() {  session.close();  sessionFactory.close();  }   @BeforeEach  void setUp() {  // Создаем и сохраняем нового клиента перед каждым тестом  Client client = new Client("Test Client", "Test Country", "123-456-7890");  clientBaseMethod.save(client);  }   @AfterEach  void tearDown() {  // Удаляем клиента после каждого теста  List<Client> clients = clientBaseMethod.find("name", "Test Client");  if (!clients.isEmpty()) {  Client client = clients.get(0);  clientBaseMethod.delete(client);  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  }   @Test  void testGet() {  List<Client> clients = clientBaseMethod.find("name", "Test Client");  assertFalse(clients.isEmpty());  Client client = clients.get(0);   Client retrievedClient = clientBaseMethod.get(client.getId());  assertNotNull(retrievedClient);  assertEquals(client.getName(), retrievedClient.getName());  }   @Test  void testFind() {  List<Client> clients = clientBaseMethod.find("name", "Test Client");  assertFalse(clients.isEmpty());  Client client = clients.get(0);   assertEquals("Test Client", client.getName());   List<Client> emptyList = clientBaseMethod.find("name", "Nonexistent Client");  assertTrue(emptyList.isEmpty());  }   @Test  void testGetAll() {  List<Client> allClients = clientBaseMethod.getAll();  assertFalse(allClients.isEmpty());  }   @Test  void testSave() {  List<Client> clients = clientBaseMethod.find("name", "Test Client");  assertFalse(clients.isEmpty());  Client client = clients.get(0);   assertEquals("Test Client", client.getName());  }   @Test  void testDelete() {  List<Client> clients = clientBaseMethod.find("name", "Test Client");  assertFalse(clients.isEmpty());  Client client = clients.get(0);   clientBaseMethod.delete(client);   List<Client> deletedClients = clientBaseMethod.find("name", "Test Client");  assertTrue(deletedClients.isEmpty());  }  }  package test;  mport org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import static org.junit.jupiter.api.Assertions.\*; import java.util.Arrays; import java.util.List;  import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import static org.junit.jupiter.api.Assertions.\*; import org.example.Store; import org.example.Client; import org.example.Invoice; public class ClientTest {   private Client client;   @BeforeEach  public void setUp() {  client = new Client();  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  @AfterEach  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  }    @Test  public void testSetName\_ValidName() {  client.setName("John");  assertEquals("John", client.getName());  }   @Test  public void testSetName\_NullName() {  assertThrows(IllegalArgumentException.class, () -> client.setName(null));  }   @Test  public void testSetName\_EmptyName() {  assertThrows(IllegalArgumentException.class, () -> client.setName(""));  }   @Test  public void testSetCountry\_ValidCountry() {  client.setCountry("USA");  assertEquals("USA", client.getCountry());  }   @Test  public void testSetCountry\_NullCountry() {  assertThrows(IllegalArgumentException.class, () -> client.setCountry(null));  }   @Test  public void testSetCountry\_EmptyCountry() {  assertThrows(IllegalArgumentException.class, () -> client.setCountry(""));  }   @Test  public void testSetPhone\_ValidPhone() {  client.setPhone("123456789");  assertEquals("123456789", client.getPhone());  }   @Test  public void testSetPhone\_NullPhone() {  assertThrows(IllegalArgumentException.class, () -> client.setPhone(null));  }   @Test  public void testSetPhone\_EmptyPhone() {  assertThrows(IllegalArgumentException.class, () -> client.setPhone(""));  }   @Test  public void testSetInvoices() {  Invoice invoice1 = new Invoice();  Invoice invoice2 = new Invoice();  client.setInvoices(Arrays.asList(invoice1, invoice2));  assertEquals(2, client.getInvoices().size());  assertTrue(client.getInvoices().contains(invoice1));  assertTrue(client.getInvoices().contains(invoice2));  }   @Test  public void testSetId() {  client.setId(1);  assertEquals(1, client.getId());  } }  package test;  import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import static org.junit.jupiter.api.Assertions.\*; import java.util.Date; import org.example.Store; import org.example.Client; import org.example.Invoice;  public class InvoiceTest {   private Invoice invoice;   @BeforeEach  public void setUp() {  invoice = new Invoice();  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  @AfterEach  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  }   @Test  public void testId() {  invoice.setId(1);  assertEquals(1, invoice.getId());  }   @Test  public void testDate() {  Date date = new Date();  invoice.setDate(date);  assertEquals(date, invoice.getDate());  }   @Test  public void testPrice() {  invoice.setPrice(100.0);  assertEquals(100.0, invoice.getPrice(), 0.001);  }   @Test  public void testStore() {  Store store = new Store();  invoice.setStore(store);  assertEquals(store, invoice.getStore());  }   @Test  public void testClient() {  Client client = new Client();  invoice.setClient(client);  assertEquals(client, invoice.getClient());  }   @Test  public void testEmptyDate() {  assertThrows(IllegalArgumentException.class, () -> invoice.setDate(null));  }   @Test  public void testNegativePrice() {  assertThrows(IllegalArgumentException.class, () -> invoice.setPrice(-10.0));  }   @Test  public void testNullStore() {  assertThrows(IllegalArgumentException.class, () -> invoice.setStore(null));  }   @Test  public void testNullClient() {  assertThrows(IllegalArgumentException.class, () -> invoice.setClient(null));  } }  package test;  import org.junit.jupiter.api.BeforeAll; import org.junit.jupiter.api.Test;  import javax.persistence.EntityManager; import javax.persistence.EntityManagerFactory; import javax.persistence.Persistence; import java.util.List;  import static org.junit.jupiter.api.Assertions.assertEquals; import static org.junit.jupiter.api.Assertions.assertNotNull;  import org.example.Invoice; import org.example.Store;  public class StoreTest {   private static EntityManagerFactory entityManagerFactory;  private static EntityManager entityManager;   @BeforeAll  public static void setUp() {  entityManagerFactory = Persistence.createEntityManagerFactory("your-persistence-unit"); // Замените на ваш UNIT  entityManager = entityManagerFactory.createEntityManager();  }   @Test  public void testStoreCreation() {  entityManager.getTransaction().begin();   // Создание и сохранение магазина  Store store = new Store();  store.setName("Магазин 1");  store.setPhone("123-456-7890");  entityManager.persist(store);   entityManager.getTransaction().commit();   // Проверка, что магазин был успешно сохранен в базе данных  Store retrievedStore = entityManager.find(Store.class, store.getId());  assertNotNull(retrievedStore);  assertEquals("Магазин 1", retrievedStore.getName());  assertEquals("123-456-7890", retrievedStore.getPhone());  }   @Test  public void testGetInvoicesForStore() {  entityManager.getTransaction().begin();   // Создание магазина  Store store = new Store();  store.setName("Магазин 2");  store.setPhone("987-654-3210");  entityManager.persist(store);   // Создание счета для магазина  Invoice invoice = new Invoice();  invoice.setStore(store);  entityManager.persist(invoice);   entityManager.getTransaction().commit();   // Получение счетов для магазина  List<Invoice> invoices = store.getInvoices();  assertNotNull(invoices);  assertEquals(1, invoices.size());  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  @AfterEach  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  } }  package test;  import org.junit.jupiter.api.BeforeAll; import org.junit.jupiter.api.Test;  import javax.persistence.EntityManager; import javax.persistence.EntityManagerFactory; import javax.persistence.Persistence; import java.util.List;  import static org.junit.jupiter.api.Assertions.assertEquals; import static org.junit.jupiter.api.Assertions.assertNotNull; import org.example.Vendor;  public class VendorTest {   private Vendor vendor;   @BeforeEach  void setUp() {  vendor = new Vendor("Country", 12345, "VendorName");  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  @AfterEach  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  }   @Test  void testGetters() {  assertEquals("Country", vendor.getCountry());  assertEquals(12345, vendor.getArticle());  assertEquals("VendorName", vendor.getName());  }   @Test  void testSetters() {  vendor.setCountry("NewCountry");  vendor.setArticle(54321);  vendor.setName("NewVendorName");   assertEquals("NewCountry", vendor.getCountry());  assertEquals(54321, vendor.getArticle());  assertEquals("NewVendorName", vendor.getName());  }   @Test  void testInvalidCountry() {  assertThrows(IllegalArgumentException.class, () -> vendor.setCountry(null));  assertThrows(IllegalArgumentException.class, () -> vendor.setCountry(""));  }   @Test  void testInvalidArticle() {  assertThrows(IllegalArgumentException.class, () -> vendor.setArticle(0));  assertThrows(IllegalArgumentException.class, () -> vendor.setArticle(-123));  }   @Test  void testInvalidName() {  assertThrows(IllegalArgumentException.class, () -> vendor.setName(null));  assertThrows(IllegalArgumentException.class, () -> vendor.setName(""));  } }  package test;  import org.junit.jupiter.api.BeforeAll; import org.junit.jupiter.api.Test;  import javax.persistence.EntityManager; import javax.persistence.EntityManagerFactory; import javax.persistence.Persistence; import java.util.List;  import static org.junit.jupiter.api.Assertions.assertEquals; import static org.junit.jupiter.api.Assertions.assertNotNull;  import org.example.Vendor; import org.example.Warehouse;  public class WarehouseTest {   private Warehouse warehouse;   @BeforeEach  void setUp() {  warehouse = new Warehouse();  }   // Закрытие EntityManager и EntityManagerFactory в конце тестов  @AfterEach  public void tearDown() {  if (entityManager != null && entityManager.isOpen()) {  entityManager.close();  }  }   @Test  void testGetters() {  warehouse.setId(1);  warehouse.setName("WarehouseName");  warehouse.setQuantity(100);  Vendor vendor = new Vendor("Country", 12345, "VendorName");  warehouse.setVendor(vendor);   assertEquals(1, warehouse.getId());  assertEquals("WarehouseName", warehouse.getName());  assertEquals(100, warehouse.getQuantity());  assertEquals(vendor, warehouse.getVendor());  }   @Test  void testSetName() {  assertDoesNotThrow(() -> warehouse.setName("ValidName"));  assertThrows(IllegalArgumentException.class, () -> warehouse.setName(null));  assertThrows(IllegalArgumentException.class, () -> warehouse.setName(""));  }   @Test  void testSetQuantity() {  assertDoesNotThrow(() -> warehouse.setQuantity(0));  assertDoesNotThrow(() -> warehouse.setQuantity(100));  assertThrows(IllegalArgumentException.class, () -> warehouse.setQuantity(-1));  }   @Test  void testSetVendor() {  Vendor vendor = new Vendor("Country", 12345, "VendorName");   assertDoesNotThrow(() -> warehouse.setVendor(vendor));  assertEquals(vendor, warehouse.getVendor());   assertDoesNotThrow(() -> warehouse.setVendor(null));  assertNull(warehouse.getVendor());  } } |
| --- |

# Список литературы

1. Васильева М.А., Хобта Д.О., Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий: Учебно-методическое пособие. Издание второе, исправленное и дополненное. М.:РУТ(МИИТ), 2023. 105–с.

2. Васильева М.А., Меркулов Д.А. Группировка и обобщение данных. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 46–с.

3. Васильева М.А., Ракинцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 63–с.

4. Балакина Е.П., Васильева М.А., Филипченко К.М. Информационное обеспечение систем управления. Методические указания к курсовому проектированию. Учебно-методическое пособие. Издание второе, исправленное и дополненное, 2023.102–с.