Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Программная инженерия»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5

по предмету “Серверное программирование”

Модели данных java,
базовые DAO классы, Criteria API.

тема

Преподаватель _____ А.А. Даничев

подпись, дата инициалы, фамилия

Студент ЗКИ21-16БВВ_031625881 18.04.2025 Е.М.Хорошко

подпись, дата инициалы, фамилия

Красноярск, 2025

СОДЕРЖАНИЕ

1. **Задание.**

В рамках данной практической работы необходимо реализовать модель данных на стороне Java для маппинга с таблицами БД. Также должны быть реализованы базовые DAO классы и расширенные методы для запросов с фильтрами, с использованием Criteria API.

2. **Ход работы.**

1. Созданы классы, описывающие все используемые таблицы в Java в пакете dao.model: Jewelry.java, JewelryType.java, JewelryMaterial.java, Material.java, Country.java, Customer.java, Order.java, OrderItem.java.

Каждый класс содержит поля, соответствующие столбцам таблиц в БД, а также аннотации JPA/Hibernate для маппинга:

- @Entity, @Table – для указания сущности и таблицы БД.
- @Id, @GeneratedValue – для первичного ключа.
- @Column – для маппинга полей.
- Аннотации связей (@OneToMany, @ManyToOne, @ManyToMany и т. д.).

Класс Jewelry (листинг кода Jewelry.java):

```
package edu.sfu.lab5.model;

import jakarta.persistence.*;
import lombok.Data;
import java.math.BigDecimal;
import java.util.Set;
@Data
@Entity
@Table(name = "jewelry")
public class Jewelry {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false, length = 100)
    private String name;

    @Column(name = "price", nullable = false, columnDefinition = "numeric(10,2)")
    private BigDecimal price;
```

```
    @Column(name = "weight", nullable = false, columnDefinition = "numeric(8,2)")
    private BigDecimal weight;

    @ManyToOne
    @JoinColumn(name = "type_id", nullable = false)
    private JewelryType type;

    @Column(name = "manufacturer", nullable = false, length = 100)
    private String manufacturer;

    @ManyToOne
    @JoinColumn(name = "country_id")
    private Country country;

    @ManyToMany(fetch = FetchType.EAGER) //
    @JoinTable(
        name = "jewelryMaterial",
        joinColumns = @JoinColumn(name = "jewelry_id"),
        inverseJoinColumns = @JoinColumn(name = "material_id")
    )
    private Set<Material> materials;

// Геттеры и сеттеры
}
```

В файл конфигурации hibernate добавлены строки привязки классов-сущностей (Приложение 1):

```
<!-- Entity mappings -->
        <mapping class="edu.sfu.lab5.model.Country"/>
        <mapping class="edu.sfu.lab5.model.JewelryType"/>
        <mapping class="edu.sfu.lab5.model.Material"/>
        <mapping class="edu.sfu.lab5.model.Jewelry"/>
        <mapping class="edu.sfu.lab5.model.JewelryMaterial"/>
        <mapping class="edu.sfu.lab5.model.Customer"/>
        <mapping class="edu.sfu.lab5.model.Order"/>
        <mapping class="edu.sfu.lab5.model.OrderItem"/>
```

2. Для каждой сущности создан DAO класс с базовыми методами CRUD и расширенными методами для запросов с использованием Criteria API (Приложение 2).

3. Методы из предыдущей работы класса TestSrvs переписаны на Criteria API (Приложение 3).

4. В классе JewelryDAO реализован метод findWithFilters() с динамическими фильтрами:

```
public List<Jewelry> findWithFilters(String name,
                                     BigDecimal minPrice,
                                     BigDecimal maxPrice,
                                     Integer typeId,
                                     String manufacturer,
                                     int firstResult, int maxResults) {
        try {
            DAO.begin();
            Session session = DAO.getSession();
            CriteriaBuilder builder = session.getCriteriaBuilder();
            CriteriaQuery<Jewelry> criteria = builder.createQuery(Jewelry.class);
            Root<Jewelry> root = criteria.from(Jewelry.class);

            List<Predicate> predicates = new ArrayList<>();

            if (name != null && !name.isEmpty()) {
                predicates.add(builder.like(
                    builder.lower(root.get("name")),
                    "%" + name.toLowerCase() + "%"
                ));
            }

            if (minPrice != null) {
                predicates.add(builder.greaterThanOrEqualTo(
                    root.get("price"),
                    minPrice
                ));
            }

            if (maxPrice != null) {
                predicates.add(builder.lessThanOrEqualTo(
                    root.get("price"),
                    maxPrice
                ));
            }

            if (typeId != null) {
                predicates.add(builder.equal(
                    root.get("type").get("id"),
                    typeId
                ));
            }
```

```
            if (manufacturer != null && !manufacturer.isEmpty()) {
                predicates.add(builder.like(
                    builder.lower(root.get("manufacturer")),
                    "%" + manufacturer.toLowerCase() + "%"
                ));
            }

            if (!predicates.isEmpty()) {
                criteria.where(builder.and(
                    predicates.toArray(new Predicate[0])
                ));
            }

            criteria.orderBy(builder.asc(root.get("name")));

            List<Jewelry> result = session.createQuery(criteria)
                .setFirstResult(firstResult)
                .setMaxResults(maxResults)
                .getResultList();
            DAO.commit();
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
```

Метод вызывается через JewelryService:

```
    public List<Jewelry> searchJewelry(String name,BigDecimal minPrice,BigDecimal maxPrice,
Integer typeId,String manufacturer) {
        try {
            DAO.begin();
            // Стандартная пагинация (первые 20 записей)
            List<Jewelry> result = jewelryDAO.findWithFilters(
                name, minPrice, maxPrice, typeId, manufacturer, 0, 20);
            DAO.commit();
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
```

Вызов метода в Main.java:

```
// 3. Сервис с фильтрами
```

```
JewelryService jewelryService = new JewelryService();

// 1: Фильтр по цене и типу (первые 20)
List<Jewelry> results1 = jewelryService.searchJewelry(
    null,
    new BigDecimal("100.00"),
    new BigDecimal("500.00"),
    1,
    null
);
System.out.println("Пример 1: Украшения типа кольцо с ценой 100-500");
jewelryService.printJewelryList(results1);

//  2: Фильтр по названию и производителю
List<Jewelry> results2 = jewelryService.searchJewelry(
    "кольцо",
    null,
    null,
    null,
    "Tiffany"
);
System.out.println("\nПример 2: Украшения с названием кольцо от Tiffany");
jewelryService.printJewelryList(results2);
```

Результаты работы метода:

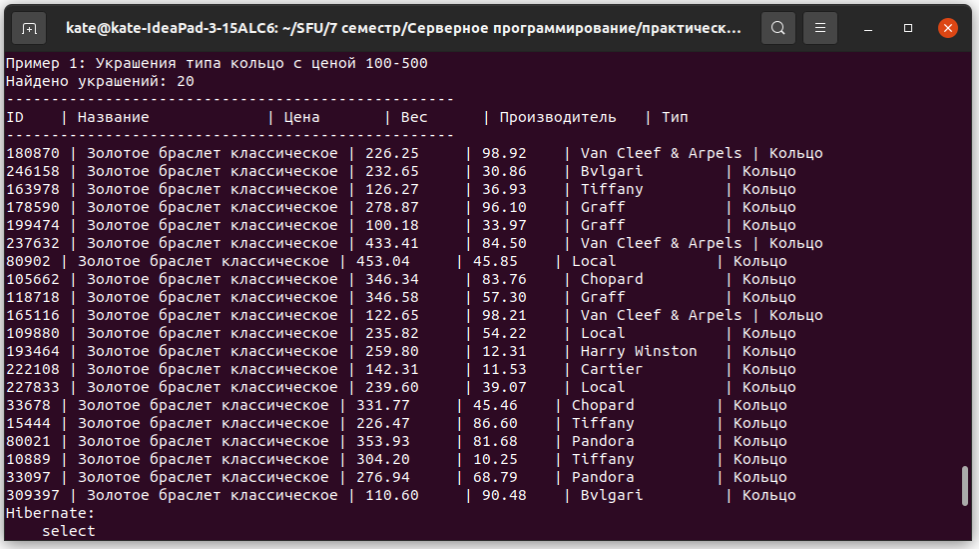

Рис.1 Метод с фильтрами - пример 1

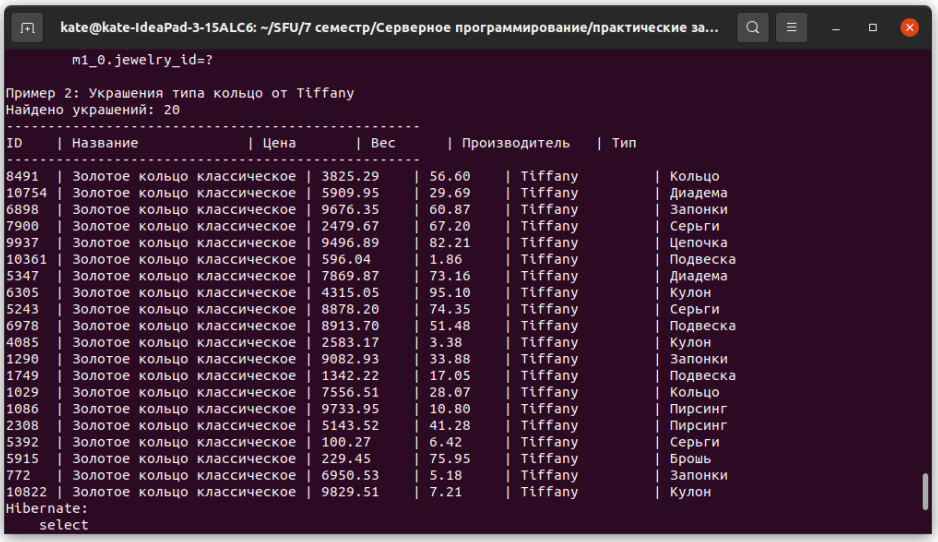

Рис.2 Метод с фильтрами - пример 2

5. Использован Criteria API для запроса, который проверяет эффективность индексов (Приложение 4). В классе JewelryDAO реализован метод findExpensiveGemstoneJewelry(int firstResult, int maxResults) с ограничением вывода в 20 записей:

```
public List<Object[]> findExpensiveGemstoneJewelry(int firstResult, int maxResults) {
        try {
            DAO.begin();
            Session session = DAO.getSession();
            CriteriaBuilder cb = session.getCriteriaBuilder();
            CriteriaQuery<Object[]> query = cb.createQuery(Object[].class);

            Root<Jewelry> jewelry = query.from(Jewelry.class);
                        Join<Jewelry, Material> materialJoin = jewelry.join("materials",
JoinType.INNER);

            Subquery<Integer> countrySubquery = query.subquery(Integer.class);
            Root<Country> country = countrySubquery.from(Country.class);
            countrySubquery.select(country.get("id"))
                           .where(country.get("name").in(Arrays.asList("Россия", "Италия",
"Франция")));

            query.multiselect(
                jewelry.get("id").alias("jewelry_id"),
                jewelry.get("name").alias("jewelry_name"),
                jewelry.get("price"),
```

```
                jewelry.get("weight"),
                materialJoin.get("name").alias("material_name"),
                materialJoin.get("type").alias("material_type"),
                materialJoin.get("carat")
            );

            List<Predicate> predicates = new ArrayList<>();
            predicates.add(cb.between(jewelry.get("price"),
                new BigDecimal("500"),
                new BigDecimal("5000")));
            predicates.add(cb.equal(materialJoin.get("type"), "gemstone"));
            predicates.add(cb.greaterThan(jewelry.get("weight"),
                new BigDecimal("10")));
            predicates.add(jewelry.get("country").get("id").in(countrySubquery));

            query.where(cb.and(predicates.toArray(new Predicate[0])));
            query.orderBy(cb.desc(jewelry.get("price")));

            List<Object[]> result = session.createQuery(query)
                .setFirstResult(firstResult)
                .setMaxResults(maxResults)
                .getResultList();
            DAO.commit();
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
```

Вызов метода в Main.java:

```
 // 5. Дорогие украшения с камнями
JewelryDAO jewelryDAO = new JewelryDAO();
List<Object[]> expensiveJewelry_page1 = jewelryDAO.findExpensiveGemstoneJewelry(0, 20);

System.out.println("\nДорогие украшения с камнями:");
System.out.println("-------------------------------------------------");
System.out.printf("%-5s | %-20s | %-10s |  %-20s%n",
                "ID", "Название", "Цена", "Материал");
System.out.println("-------------------------------------------------");
            for (Object[] row : expensiveJewelry_page1) {
                System.out.printf("%-5d | %-20s | %-10.2f | %-20s%n",
                    row[0], row[1], row[2], row[4]);
            }
```

Результаты работы метода:

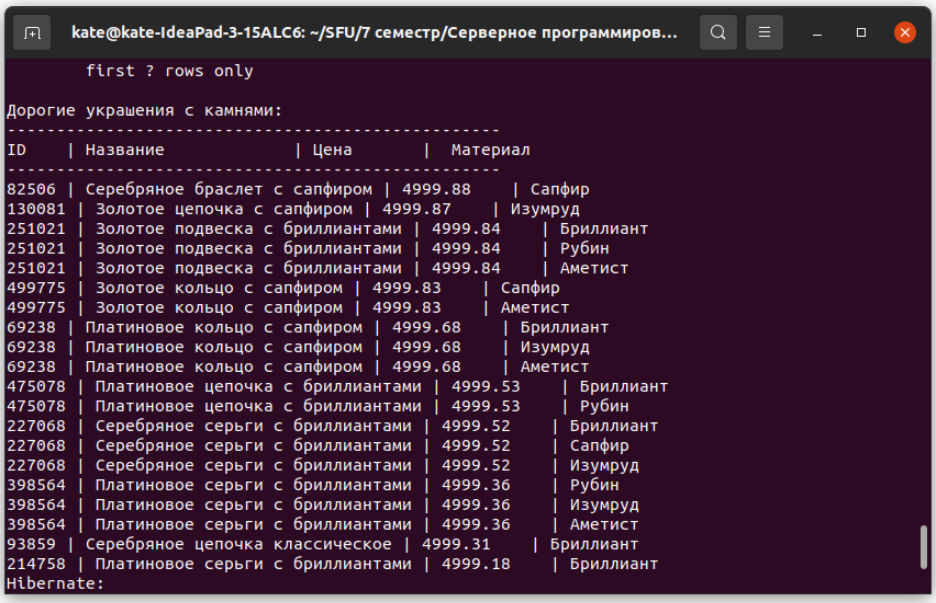

Рис.3 Дорогие украшения с камнями

6. Реализация CRUD операций: добавлены методы для работы с сущностями в рамках сессии. В качестве примера DAO класс сущности Jewelry.

JewelryDAO.java (листинг кода):

```
// CRUD операции
    public Jewelry getById(Integer id) {
        try {
            DAO.begin();
            Jewelry jewelry = DAO.getSession().get(Jewelry.class, id);
            DAO.commit();
            return jewelry;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    public List<Jewelry> getAll() {
        try {
            DAO.begin();
            List<Jewelry> result = DAO.getSession()
                .createQuery("FROM Jewelry", Jewelry.class)
                .getResultList();
            DAO.commit();
```

```
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    @SuppressWarnings("deprecation")
        public Integer save(Jewelry jewelry) {
        try {
            DAO.begin();
            Integer id = (Integer) DAO.getSession().save(jewelry);
            DAO.commit();
            return id;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    public void update(Jewelry jewelry) {
        try {
            DAO.begin();
            DAO.getSession().merge(jewelry);
            DAO.commit();
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    public void delete(Integer id) {
        try {
            DAO.begin();
            Jewelry jewelry = DAO.getSession().get(Jewelry.class, id);
            if (jewelry != null) {
                DAO.getSession().remove(jewelry);
            }
            DAO.commit();
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    @SuppressWarnings("deprecation")
        public void saveOrUpdate(Jewelry jewelry) {
        try {
            DAO.begin();
```

```
            DAO.getSession().saveOrUpdate(jewelry);
            DAO.commit();
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
```

7. Для демонстрации работы приложения с реализацией связи один-ко-многим был создан сервисный класс DemoService, содержащий метод demonstrateOneToMany() (Приложение 5). В качестве примера использована связь между сущностями Country (один) и Jewelry (много), где одна страна может производить множество ювелирных изделий. Инициализация данных реализована в методе initializeReferenceData():

- Создание страны (Россия) при отсутствии
- Создание типа украшения (Кольцо)
- Создание материалов (Золото, Бриллиант)

Для фильтрации по стране (WHERE country_id = 1), сортировки по ID (ORDER BY id) и ограничения количества результатов (LIMIT 10) использовалось Criteria API. Все операции выполняются в транзакции и при ошибке выполняется откат (rollback). Результат работы метода выводится в консоль с форматированным выводом.

Данная реализация наглядно демонстрирует работу связи один-ко-многим между сущностями и возможности Criteria API для построения сложных запросов с фильтрацией и ограничением результатов.

Результат работы демо-сервиса:

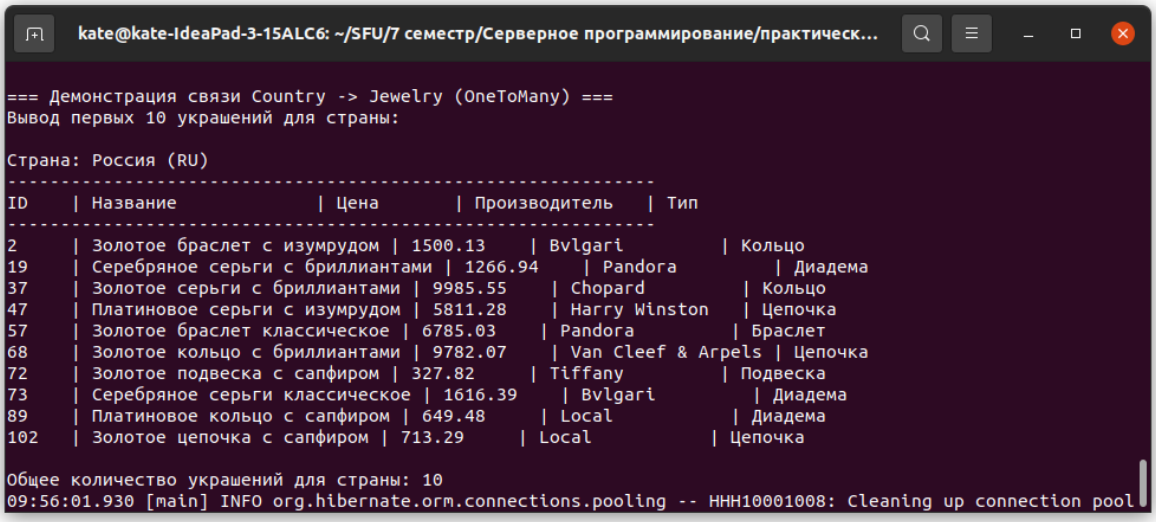

Рис.4 Демонстрация связи OneToMany

Листинг кода hibernate.cfg.xml:

```
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
       <!-- Database connection -->
       <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
                                                                                <property
name="hibernate.connection.url">jdbc:postgresql://localhost:5432/jewelry_dbg</property>
       <property name="hibernate.connection.username">postgres</property>
       <property name="hibernate.connection.password">postgres</property>
                                                                                <property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
       <property name="hibernate.show_sql">true</property>
       <property name="hibernate.format_sql">true</property>

       <!-- Entity mappings -->
       <mapping class="edu.sfu.lab5.model.Country"/>
       <mapping class="edu.sfu.lab5.model.JewelryType"/>
       <mapping class="edu.sfu.lab5.model.Material"/>
       <mapping class="edu.sfu.lab5.model.Jewelry"/>
       <mapping class="edu.sfu.lab5.model.JewelryMaterial"/>
       <mapping class="edu.sfu.lab5.model.Customer"/>
       <mapping class="edu.sfu.lab5.model.Order"/>
       <mapping class="edu.sfu.lab5.model.OrderItem"/>
   </session-factory>
</hibernate-configuration>
```

Листинг кода DAO.java:

```
package edu.sfu.lab5.manager;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
public class DAO {
   //private static final SessionFactory sessionFactory = buildSessionFactory();
       private static final SessionFactory sessionFactory;
   private static final ThreadLocal<Session> currentSession = new ThreadLocal<>();
   private static final ThreadLocal<Transaction> currentTransaction = new ThreadLocal<>();
   static {
       try {
           StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
               .configure() // loads hibernate.cfg.xml
               .build();

           sessionFactory = new MetadataSources(registry)
               .addAnnotatedClass(edu.sfu.lab5.model.Country.class)
               .addAnnotatedClass(edu.sfu.lab5.model.JewelryType.class)
               // добавьте все остальные сущности
               .buildMetadata()
               .buildSessionFactory();
       } catch (Exception ex) {
           System.err.println("Initial SessionFactory creation failed: " + ex);
           throw new ExceptionInInitializerError(ex);
       }
   }

   public static Session getSession() {
       Session session = currentSession.get();
       if (session == null || !session.isOpen()) {
           session = sessionFactory.openSession();
           currentSession.set(session);
       }
       return session;
   }
```

```
    public static void begin() {
        Transaction transaction = currentTransaction.get();
        if (transaction == null) {
            transaction = getSession().beginTransaction();
            currentTransaction.set(transaction);
        }
    }
    public static void commit() {
        Transaction transaction = currentTransaction.get();
        if (transaction != null && transaction.isActive()) {
            transaction.commit();
            currentTransaction.remove();
            closeSession();
        }
    }
    public static void rollback() {
        Transaction transaction = currentTransaction.get();
        if (transaction != null && transaction.isActive()) {
            transaction.rollback();
            currentTransaction.remove();
            closeSession();
        }
    }
    public static void closeSession() {
        Session session = currentSession.get();
        if (session != null && session.isOpen()) {
            session.close();
            currentSession.remove();
        }
    }
    public static void shutdown() {
        sessionFactory.close();
    }
}
```

Листинг кода TestSrvs.java:

```
package edu.sfu.lab5.services;
import edu.sfu.lab5.manager.DAO;
import edu.sfu.lab5.model.Country;
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import jakarta.persistence.criteria.Root;
import jakarta.persistence.metamodel.EntityType;
import jakarta.persistence.metamodel.Metamodel;
import java.util.List;
import org.hibernate.Session;
public class TestSrvs {

    // Получение имени страны по ID с использованием Criteria API
    public String getName(int id) {
        try {
            DAO.begin();
            Session session = DAO.getSession();

            // Проверка метамодели
            Metamodel metamodel = session.getMetamodel();
            EntityType<Country> countryEntity = metamodel.entity(Country.class);
            System.out.println("Country entity detected: " + countryEntity);

            CriteriaBuilder builder = session.getCriteriaBuilder();
            CriteriaQuery<String> criteria = builder.createQuery(String.class);
            Root<Country> root = criteria.from(Country.class);

            criteria.select(root.get("name"))
                    .where(builder.equal(root.get("id"), id));

            String result = session.createQuery(criteria).uniqueResult();
            DAO.commit();
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    // Получение списка имен в диапазоне ID с использованием Criteria API
    public List<String> getNamesInRange(int startId, int endId) {
        try {
            DAO.begin();
```

```
            Session session = DAO.getSession();
            CriteriaBuilder builder = session.getCriteriaBuilder();

            CriteriaQuery<String> criteria = builder.createQuery(String.class);
            Root<Country> root = criteria.from(Country.class);

            criteria.select(root.get("name"))
                    .where(builder.between(root.get("id"), startId, endId));

            List<String> result = session.createQuery(criteria).getResultList();
            DAO.commit();
            return result;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    // Создание новой записи
    public int createCountry(String name, String code) {
        try {
            DAO.begin();
            Session session = DAO.getSession();

            Country country = new Country();
            country.setName(name);
            country.setCode(code);

            session.persist(country);
            DAO.commit();
            return 1;
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
}
```

Лиснинг SQL запроса:

```
SELECT
	j.id AS jewelry_id,
	j.name AS jewelry_name,
	j.price,
	j.weight,
	m.name AS material_name,
	m.type AS material_type,
	m.carat
FROM
	Jewelry j
JOIN
	JewelryMaterial jm ON j.id = jm.jewelry_id
JOIN
	Material m ON jm.material_id = m.id
WHERE
	j.price BETWEEN 500 AND 5000
	AND m.type = 'gemstone'
	AND j.weight > 10
	AND j.country_id IN (
	SELECT id FROM Country
	WHERE name IN ('Россия', 'Италия', 'Франция')
	)
ORDER BY
	j.price DESC
LIMIT 20;
```

Листинг кода DemoService.java:

```
package edu.sfu.lab5.services;
import edu.sfu.lab5.dao.*;
import edu.sfu.lab5.manager.DAO;
import edu.sfu.lab5.model.*;
import java.math.BigDecimal;
import java.util.*;
public class DemoService {
   private final CountryDAO countryDAO = new CountryDAO();
   private final JewelryDAO jewelryDAO = new JewelryDAO();
   private final JewelryTypeDAO jewelryTypeDAO = new JewelryTypeDAO();
   private final MaterialDAO materialDAO = new MaterialDAO();
   public void demonstrateOneToMany(int limit) {
       try {
           DAO.begin();
             // 1. Загружаем или создаем необходимые справочные данные
              initializeReferenceData();
              // 2. Получаем и выводим первые N записей
              List<Jewelry> jewelryList = jewelryDAO.findFirstNByCountryId(1, limit);
              printResults(jewelryList);
              DAO.commit();
       } catch (Exception e) {
           DAO.rollback();
           throw e;
       }

   }


   private void initializeReferenceData() {
        try {
            DAO.begin();
              // Создаем страну, если не существует
              if (countryDAO.getById(1) == null) {
                  Country russia = new Country();
                  russia.setId(1);
                  russia.setName("Россия");
                  russia.setCode("RU");
                  countryDAO.save(russia);
              }
```

```
            // Создаем тип украшения, если не существует
            if (jewelryTypeDAO.getById(1) == null) {
                JewelryType ringType = new JewelryType();
                ringType.setId(1);
                ringType.setName("Кольцо");
                jewelryTypeDAO.save(ringType);
            }

            // Создаем материалы, если не существуют
            if (materialDAO.getById(1) == null) {
                Material gold = new Material();
                gold.setId(1);
                gold.setName("Золото");
                gold.setType("metal");
                gold.setCarat(new BigDecimal("24.00"));
                materialDAO.save(gold);
            }

            if (materialDAO.getById(2) == null) {
                Material diamond = new Material();
                diamond.setId(2);
                diamond.setName("Бриллиант");
                diamond.setType("gemstone");
                diamond.setCarat(new BigDecimal("1.00"));
                diamond.setQuality("VS1");
                materialDAO.save(diamond);
            }
            DAO.commit();
        } catch (Exception e) {
            DAO.rollback();
            throw e;
        }
    }
    private void printResults(List<Jewelry> jewelryList) {
        System.out.println("\n=== Демонстрация связи Country -> Jewelry (OneToMany) ===");
        System.out.println("Вывод первых " + jewelryList.size() + " украшений для страны:");

        if (!jewelryList.isEmpty()) {
            Country country = jewelryList.get(0).getCountry();
            System.out.println("\nСтрана: " + country.getName() + " (" + country.getCode() +
")");
        }
        System.out.println("-------------------------------------------------------------");
        System.out.printf("%-5s | %-20s | %-10s | %-15s | Тип%n",
            "ID", "Название", "Цена", "Производитель");
```

```
        System.out.println("-----------------------------------------------------------------");
        for (Jewelry jewelry : jewelryList) {
            System.out.printf("%-5d | %-20s | %-10.2f | %-15s | %s%n",
                jewelry.getId(),
                jewelry.getName(),
                jewelry.getPrice(),
                jewelry.getManufacturer(),
                jewelry.getType().getName());
        }

        System.out.println("\nОбщее количество украшений для страны: " +
jewelryList.size());
    }
}
```