МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Базы данных»

Тема: Нагрузочное тестирование БД

Студент гр. 3384 _____ Козьмин Н.В.

Преподаватель _____ Михайлова С.В.

Санкт-Петербург

2025

Цель работы.

- Написать скрипт, заполняющий БД большим количеством тестовых данных
- Измерить время выполнения запросов, написанных в ЛР3
    - Проверить для числа записей:
        - 100 записей в каждой табличке
        - 1.000 записей
        - 10.000 записей
        - 1.000.000 записей
        - можно больше.
    - Все запросы выполнять с фиксированным ограничением на вывод (LIMIT), т.к. запросы без LIMIT всегда будет выполняться O(n) от кол-ва записей
    - Проверить влияние сортировки на скорость выполнения запросов.
    - Для измерения использовать фактическое (не процессорное и т.п.) время
- Добавить в БД индексы (хотя бы 5 штук). Измерить влияние (или его отсутствие) индексов на скорость выполнения запросов. Обратите внимание на:
    - Скорость сортировки больших табличек
    - Скорость JOIN (но остальные запросы тоже требуется проверить)

Задание.

Вариант 8. Пусть требуется создать программную систему, предназначенную для директора продовольственного магазина. Такая система должна обеспечивать хранение сведений о магазине, об имеющихся в нем товарах, о торговых базах и товарах, хранящихся на этих базах. Магазин осуществляет закупку товаров на разных базах, предпочитая при этом закупать одни виды товара на одних базах, а другие на других. Магазин характеризуется классом, номером и имеет несколько отделов. Каждый товар в каждом магазине

продается, по крайней мере, в одном отделе. Каждый отдел имеет заведующего. Товары, имеющиеся в магазине и хранящиеся на базах, характеризуются ценой, сортом и количеством. Розничные цены в магазине зависят от класса магазина. Директор магазина должен иметь возможность изменить цену товара по своему усмотрению, осуществить закупку недостающего товара на базе. Он может также закрыть один из отделов или открыть новый, при этом товары могут перемещаться из отдела в отдел. Директору могут потребоваться следующие сведения:

- Какие товары имеются в магазине (на базе)?
- Какие отсутствующие товары может заказать магазин на базе?
- Какие товары, и в каком количестве имеются в отделе магазина?
- Список заведующих отделами магазина?
- Суммарная стоимость товара в каждом отделе?
- На каких базах, и в каких количествах есть товар нужного наименования?

Выполнение.

Напишем скрипт populate_test_data_faker, который добавляет n записей (или при необходимости больше) с данными, похожими на правдивые, в каждую таблицу.

Запросы разделим на отдельные функции с необходимыми опциями и show_output для логов, а также добавив обязательные подсчеты времени выполнения. Затем сделаем объединяющую функцию, которая вызывает все остальные с переданным значением show_output.

Добавим требуемый LIMIT на запросы. Сортировки на время проверок будем комментировать. Проведем первые запуски и отобразим результаты см. таблица 1 и 2.

Таблица 1 – Время выполнения запросов [сек] от количества записей без сортировки

	add	1.1	1.2	2.1	2.2	3	4	5	6
100	0.39	0.0075	0.0051	0.0085	0.0105	0.0020	0.0021	0.0045	0.0058
1000	2.81	0.0063	0.0025	0.0060	0.0045	0.0030	0.0020	0.0095	0.0043
10000	150	0.0076	0.0023	0.0076	0.0077	0.0020	0.0035	0.3846	0.0075
20000	569	0.0174	0.0035	0.0113	0.0090	0.0010	0.0053	0.2009	0.0145

Таблица 2 – Время выполнения запросов [сек] от количества записей с сортировкой

	add	1.1	1.2	2.1	2.2	3	4	5	6
100	0.43	0.0064	0.0035	0.0092	0.0062	0.0013	0.0023	0.0052	0.0032
1000	2.82	0.0065	0.0034	0.0050	0.0057	0.0030	0.0020	0.0090	0.0030
10000	140	0.0070	0.0030	0.0075	0.0070	0.0023	0.0030	0.3406	0.0090
20000	595	0.0103	0.0030	0.0092	0.0154	0.0020	0.0030	1.0122	0.0917

Теперь добавим индексы и проведем те же самые эксперименты. Для этого включим их создание через функцию create_indexes в populate_data.py с проверкой на отсутствие (IF NOT EXISTS) для последующих запусков.

Таблица 3 – Время выполнения запросов [сек] от количества записей без сортировки и с добавлением индексов

	add	1.1	1.2	2.1	2.2	3	4	5	6
100	0.33	0.0060	0.0027	0.0070	0.0050	0.0030	0.0030	0.0039	0.0022
1000	2.78	0.0075	0.0025	0.0068	0.0060	0.0020	0.0021	0.0085	0.0037
10000	138	0.0064	0.0020	0.0085	0.0064	0.0021	0.0020	0.3172	0.0155
20000	558	0.0080	0.0033	0.0072	0.0080	0.0034	0.0020	0.1572	0.0093

Таблица 4 – Время выполнения запросов [сек] от количества записей с сортировкой и с добавлением индексов

	add	1.1	1.2	2.1	2.2	3	4	5	6
100	0.35	0.0058	0.0032	0.0095	0.0060	0.0024	0.0020	0.0061	0.0020
1000	2.83	0.0060	0.0028	0.0070	0.0060	0.0020	0.0020	0.0255	0.0034
10000	138	0.0060	0.0030	0.0065	0.0065	0.0020	0.0020	0.3478	0.0085
20000	572	0.0080	0.0030	0.0071	0.0090	0.0021	0.0015	0.8079	0.0136

Видно, что с добавление индексов даже для добавления записей (add) скорость выполнения выше. Чем больше добавляемых элементов, тем больше разница. Отсортированные же данные добавляются с большими задержками, что очевидно, а разница по времени выполнения запросов ощутимо падает при 5ом и 6ом запросе. В 5ом запросе это обусловлено лишними дополнительными вычислениями, а 6ой запрос хорошо оптимизируется индексами, на что также хорошо влияет в купе заблаговременная сортировка, так как нужные индексы находятся в одном месте. Приведем ссылки на результаты работы в приложении А. Исходный код со см. в приложении Б.

Выводы.

В ходе выполнения лабораторной работы изучены новые методы взаимодействия с бд, используемые для нагрузочного тестирования. В том числе задействованы методы автоматического заполнения данными (похожими на правду) и методы индексирования для оптимизации. Дополнительно проделан анализ того, как влияют сортированные и несортированные данные на добавление данных и на запросы с индексами и без, что помогло составить полную картину о результативности перечисленных методов. Перечисленные требования выполнены и модернизированная программа отрабатывает успешно.

ПРИЛОЖЕНИЕ А
ССЫЛКИ НА РЕЗУЛЬТАТЫ

Ссылка на pr:

https://github.com/moevm/sql-2025-3384/pull/42

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД

main.py:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base
from populate_data import populate_test_data_faker, create_indexes, drop_indexes
from queries import *
import os

def main():
    # Параметры подключения к PostgreSQL
    db_host = os.getenv('DB_HOST', 'localhost')
    db_port = os.getenv('DB_PORT', '5432')
    db_name = os.getenv('DB_NAME', 'university_store_database')
    db_user = os.getenv('DB_USER', 'postgres')
    db_password = os.getenv('DB_PASSWORD', 'postgres')

    # Создание подключения к PostgreSQL
    database_url = f'postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}'
    engine = create_engine(
        database_url,
        echo=False,
    )

    # Создание таблиц
    print("Создание таблиц в PostgreSQL...")
    Base.metadata.create_all(engine)

    # Создание сессии
    Session = sessionmaker(bind=engine)
    session = Session()

    try:
        create_indexes(engine)

        # Заполнение тестовыми данными
        populate_test_data_faker(session, records_per_table=1000)

        execute_queries(session, show_output=False)

    except Exception as e:
        print(f"Произошла ошибка: {e}")
        session.rollback()
    finally:
        session.close()

if __name__ == "__main__":
    main()
```

models.py (без изменений с предыдущей лаб):

```
from sqlalchemy import create_engine, Column, Integer, String, Text, ForeignKey, DECIMAL
from sqlalchemy.ext.declarative import declarative_base
```

```
    from sqlalchemy.orm import relationship, sessionmaker

    Base = declarative_base()

    class StoreClass(Base):
        __tablename__ = 'store_class'

        store_class_id = Column(Integer, primary_key=True)
        name = Column(String(50), nullable=False)
        description = Column(Text)

        stores = relationship("Store", back_populates="store_class")
        product_prices = relationship("ProductPrice",
back_populates="store_class")

    class TradingBase(Base):
        __tablename__ = 'trading_base'

        trading_base_id = Column(Integer, primary_key=True)
        name = Column(String(100), nullable=False)
        description = Column(Text)

        warehouse_products = relationship("WarehouseProduct",
back_populates="trading_base")
        warehouse_priorities = relationship("WarehousePriority",
back_populates="trading_base")

    class Employee(Base):
        __tablename__ = 'employee'

        employee_id = Column(Integer, primary_key=True)
        first_name = Column(String(50), nullable=False)
        last_name = Column(String(50), nullable=False)

        managed_stores = relationship("Store", back_populates="director")
        managed_departments = relationship("Department",
back_populates="manager")

    class Store(Base):
        __tablename__ = 'store'

        store_id = Column(Integer, primary_key=True)
        name = Column(String(100), nullable=False)
        description = Column(Text)
        store_class_id = Column(Integer,
ForeignKey('store_class.store_class_id'))
        director_id = Column(Integer, ForeignKey('employee.employee_id'))

        store_class = relationship("StoreClass", back_populates="stores")
        director = relationship("Employee", back_populates="managed_stores")
        departments = relationship("Department", back_populates="store")
        warehouse_priorities = relationship("WarehousePriority",
back_populates="store")

    class Department(Base):
        __tablename__ = 'department'

        department_id = Column(Integer, primary_key=True)
        store_id = Column(Integer, ForeignKey('store.store_id'))
        manager_id = Column(Integer, ForeignKey('employee.employee_id'))
        name = Column(String(100), nullable=False)

        store = relationship("Store", back_populates="departments")
```

```
        manager                    =                    relationship("Employee",
back_populates="managed_departments")
        department_products        =        relationship("DepartmentProduct",
back_populates="department")

    class Product(Base):
        __tablename__ = 'product'

        article = Column(Integer, primary_key=True)
        name = Column(String(100), nullable=False)
        sort = Column(String(50))

        department_products        =        relationship("DepartmentProduct",
back_populates="product")
        warehouse_products         =         relationship("WarehouseProduct",
back_populates="product")
        product_prices = relationship("ProductPrice", back_populates="product")
        warehouse_priorities       =        relationship("WarehousePriority",
back_populates="product")

    class DepartmentProduct(Base):
        __tablename__ = 'department_product'

        department_id = Column(Integer, ForeignKey('department.department_id'),
primary_key=True)
        article     =     Column(Integer,     ForeignKey('product.article'),
primary_key=True)
        count = Column(Integer, default=0)

        department                 =                relationship("Department",
back_populates="department_products")
        product = relationship("Product", back_populates="department_products")

    class WarehouseProduct(Base):
        __tablename__ = 'warehouse_product'

        trading_base_id                    =                    Column(Integer,
ForeignKey('trading_base.trading_base_id'), primary_key=True)
        article     =     Column(Integer,     ForeignKey('product.article'),
primary_key=True)
        count = Column(Integer, default=0)
        price = Column(DECIMAL(10, 2), nullable=False)

        trading_base               =               relationship("TradingBase",
back_populates="warehouse_products")
        product = relationship("Product", back_populates="warehouse_products")

    class ProductPrice(Base):
        __tablename__ = 'product_price'

        store_class_id                     =                    Column(Integer,
ForeignKey('store_class.store_class_id'), primary_key=True)
        article     =     Column(Integer,     ForeignKey('product.article'),
primary_key=True)
        price = Column(DECIMAL(10, 2), nullable=False)

        store_class                =                relationship("StoreClass",
back_populates="product_prices")
        product = relationship("Product", back_populates="product_prices")

    class WarehousePriority(Base):
        __tablename__ = 'warehouse_priority'
```

```
        article = Column(Integer, ForeignKey('product.article'),
primary_key=True)
        store_id = Column(Integer, ForeignKey('store.store_id'),
primary_key=True)
        trading_base_id = Column(Integer,
ForeignKey('trading_base.trading_base_id'), primary_key=True)
        priority = Column(Integer, default=5)

        product = relationship("Product",
back_populates="warehouse_priorities")
        store = relationship("Store", back_populates="warehouse_priorities")
        trading_base = relationship("TradingBase",
back_populates="warehouse_priorities")
```

populate_data.py:

```
    from sqlalchemy import create_engine, text
    from sqlalchemy.orm import sessionmaker
    from models import Base, StoreClass, TradingBase, Employee, Store,
Department, Product, DepartmentProduct, WarehouseProduct, ProductPrice,
WarehousePriority
    from faker import Faker
    import time
    from datetime import datetime


    truncate_sql = """
    TRUNCATE TABLE
        department_product,
        department,
        warehouse_priority,
        store,
        product_price,
        store_class,
        warehouse_product,
        trading_base,
        employee,
        product
    RESTART IDENTITY;
    """


    def populate_test_data_faker(session, records_per_table=10,
use_truncate=True):
        start_time = time.time()
        fake = Faker('ru_RU')  # Русская локализация

        # Очистка существующих данных
        if use_truncate:
            print("Очистка существующих данных...")
            session.execute(text(truncate_sql))
            session.commit()

        print(f"Генерация тестовых данных ({records_per_table} записей на
таблицу)...")


        # print("Генерация классов магазинов...")
        store_classes_data = [
            ('Элитный', 'Магазины премиум-класса с широким ассортиментом'),
            ('Стандарт', 'Магазины среднего ценового сегмента'),
            ('Эконом', 'Бюджетные магазины'),
            ('Специализированный', 'Магазины узкой направленности'),
```

```
            ('Гипермаркет', 'Крупные торговые центры')
        ]

        store_classes = []
        for i in range(min(records_per_table, len(store_classes_data))):
            if i < len(store_classes_data):
                name, description = store_classes_data[i]
            else:
                name = fake.company()
                description = fake.text(max_nb_chars=100)

            store_classes.append(StoreClass(
                name=name,
                description=description
            ))

        session.add_all(store_classes)
        session.flush()
        print(f"Создано {len(store_classes)} классов магазинов")


        # print("Генерация торговых баз...")
        trading_bases = []
        for i in range(records_per_table):
            trading_bases.append(TradingBase(
                name=fake.company() + ' ' + fake.random_element(['Центр',
'База', 'Склад', 'Комплекс', 'Хаб']),
                description=fake.text(max_nb_chars=150)
            ))

        session.add_all(trading_bases)
        session.flush()
        print(f"Создано {len(trading_bases)} торговых баз")


        # print("Генерация сотрудников...")
        employees = []
        for i in range(records_per_table * 2):   # Больше сотрудников для
менеджеров
            employees.append(Employee(
                first_name=fake.first_name(),
                last_name=fake.last_name()
            ))

        session.add_all(employees)
        session.flush()
        print(f"Создано {len(employees)} сотрудников")


        # print("Генерация магазинов...")
        stores = []
        store_names = ['Супермаркет', 'Гипермаркет', 'Магазин', 'Торговый
центр', 'Универмаг', 'Маркет', 'Торговая точка']

        for i in range(records_per_table):
            stores.append(Store(
                name=fake.random_element(store_names) + " " + fake.company(),
                description=fake.text(max_nb_chars=20),
                store_class_id=fake.random_element([sc.store_class_id for sc in
store_classes]),
                director_id=fake.random_element([e.employee_id for e in
employees])
            ))
```

```
        session.add_all(stores)
        session.flush()
        print(f"Создано {len(stores)} магазинов")


        # print("Генерация отделов...")
        departments = []
        department_names = [
            'Молочный отдел', 'Хлебный отдел', 'Овощной отдел', 'Мясной отдел',
'Бакалея',
            'Гастрономия',    'Кондитерский    отдел',    'Бытовая    техника',
'Электроника', 'Напитки',
            'Фруктовый  отдел',  'Рыбный  отдел',  'Колбасный  отдел',  'Сыры',
'Замороженные продукты',
            'Хозтовары', 'Косметика', 'Одежда', 'Обувь', 'Аксессуары'
        ]

        # Обязательное количество отделов для каждого магазина
        for s_id in map(lambda store: store.store_id, stores):
            departments.append(Department(
                store_id=s_id,
                manager_id=fake.random_element([e.employee_id    for    e    in
employees]),
                name=fake.random_element(department_names) + (' ' + fake.word()
if fake.boolean() else '')
            ))
        # Дополнительное количество отделов для каждого магазина
        for i in range(records_per_table):
            departments.append(Department(
                store_id=fake.random_element([s.store_id for s in stores]),
                manager_id=fake.random_element([e.employee_id    for    e    in
employees]),
                name=fake.random_element(department_names) + (' ' + fake.word()
if fake.boolean() else '')
            ))

        session.add_all(departments)
        session.flush()
        print(f"Создано {len(departments)} отделов")


        # print("Генерация товаров...")
        products = []
        product_types = {
            'Молочные':   ['Молоко',   'Йогурт',   'Сыр',   'Кефир',   'Творог',
'Сметана'],
            'Хлебные': ['Хлеб', 'Булочка', 'Батон', 'Пирог', 'Печенье'],
            'Овощи': ['Картофель', 'Морковь', 'Лук', 'Помидоры', 'Огурцы'],
            'Мясо': ['Курица', 'Говядина', 'Свинина', 'Баранина', 'Колбаса'],
            'Бакалея': ['Рис', 'Гречка', 'Макароны', 'Мука', 'Сахар'],
            'Напитки': ['Сок', 'Вода', 'Лимонад', 'Чай', 'Кофе'],
            'Электроника':  ['Телевизор',  'Смартфон',  'Ноутбук',  'Планшет',
'Наушники']
        }

        for i in range(records_per_table):
            category = fake.random_element(list(product_types.keys()))
            product_name = fake.random_element(product_types[category])

            products.append(Product(
                name=f"{product_name} {fake.word()}",
```

```
                sort=fake.random_element(['Премиум', 'Стандарт', 'Эконом'] +
(['Высший сорт', 'Первый сорт', 'Отборный'] if category != 'Электроника' else []))
            ))

        session.add_all(products)
        session.flush()
        print(f"Создано {len(products)} товаров")


        # print("Генерация товаров в отделах...")
        department_products = []

        # Создаем связи между отделами и товарами
        used_combinations = set()
        for i in range(records_per_table * 5):
            dept = fake.random_element(departments)
            prod = fake.random_element(products)

            # Избегаем дубликатов
            combination = (dept.department_id, prod.article)
            if combination not in used_combinations:
                department_products.append(DepartmentProduct(
                    department_id=dept.department_id,
                    article=prod.article,
                    count=fake.random_int(min=0, max=100)
                ))
                used_combinations.add(combination)

        session.add_all(department_products)
        session.flush()
        print(f"Создано {len(department_products)} связей товаров с отделами")


        # print("Генерация товаров на складах...")
        warehouse_products = []

        used_combinations = set()
        for i in range(records_per_table * 2):
            base = fake.random_element(trading_bases)
            prod = fake.random_element(products)

            combination = (base.trading_base_id, prod.article)
            if combination not in used_combinations:
                warehouse_products.append(WarehouseProduct(
                    trading_base_id=base.trading_base_id,
                    article=prod.article,
                    count=fake.random_int(min=10, max=500),
                    price=float(fake.random_int(min=50, max=50000) / 100)   #
Цены от 0.50 до 500.00
                ))
                used_combinations.add(combination)

        session.add_all(warehouse_products)
        session.flush()
        print(f"Создано {len(warehouse_products)} позиций на складах")


        # print("Генерация цен товаров...")
        product_prices = []
        for store_class in store_classes:
            for prod in products:
                base_price = float(fake.random_int(min=100, max=10000) / 100)
                if store_class.name == 'Элитный':
```

```
                    price = base_price * 1.3  # На 30% дороже
                elif store_class.name == 'Эконом':
                    price = base_price * 0.8  # На 20% дешевле
                else:
                    price = base_price

                product_prices.append(ProductPrice(
                    store_class_id=store_class.store_class_id,
                    article=prod.article,
                    price=price
                ))
        session.add_all(product_prices)
        session.flush()
        print(f"Создано {len(product_prices)} ценовых позиций")


        # print("Генерация приоритетов поставок...")
        warehouse_priorities = []

        used_combinations = set()
        for i in range(records_per_table * 2):
            prod = fake.random_element(products)
            store = fake.random_element(stores)
            base = fake.random_element(trading_bases)

            combination = (prod.article, store.store_id, base.trading_base_id)
            if combination not in used_combinations:
                warehouse_priorities.append(WarehousePriority(
                    article=prod.article,
                    store_id=store.store_id,
                    trading_base_id=base.trading_base_id,
                    priority=fake.random_int(min=1, max=10)
                ))
                used_combinations.add(combination)

        session.add_all(warehouse_priorities)
        session.flush()
        print(f"Создано {len(warehouse_priorities)} приоритетов поставок")

        # Фиксация изменений
        session.commit()

        end_time = time.time()
        execution_time = end_time - start_time

        print(f"Время выполнения: {execution_time:.2f} секунд")


    def create_indexes(engine):
        print("Создание индексов для улучшения производительности...")

        indexes_sql = [
            # 1. Индекс для поиска по названию товара
            "CREATE INDEX IF NOT EXISTS idx_product_name_search ON product
(name);",

            # 2. Индекс для JOIN между Department и Store
            "CREATE INDEX IF NOT EXISTS idx_department_store_id_fk ON department
(store_id);",

            # 3. Индексы для сортировки по количеству
            "CREATE INDEX IF NOT EXISTS idx_department_product_count_sort ON
department_product (count);",
```

```
            "CREATE INDEX IF NOT EXISTS idx_warehouse_product_count_sort ON
warehouse_product (count);",

            # 4. Индекс для сортировки по приоритетам
            "CREATE INDEX IF NOT EXISTS idx_warehouse_priority_sort ON
warehouse_priority (priority);",

            # 5. Индексы для сортировки по ценам
            "CREATE INDEX IF NOT EXISTS idx_product_price_sort ON product_price
(price);",
            "CREATE INDEX IF NOT EXISTS idx_warehouse_product_price_sort ON
warehouse_product (price);",

            # 6. Составной индекс для DepartmentProduct
            "CREATE INDEX IF NOT EXISTS idx_department_product_join ON
department_product (department_id, article);",

            # 7. Составной индекс для WarehouseProduct
            "CREATE INDEX IF NOT EXISTS idx_warehouse_product_join ON
warehouse_product (trading_base_id, article);",

            # 8. Составной индекс для ProductPrice
            "CREATE INDEX IF NOT EXISTS idx_product_price_join ON product_price
(store_class_id, article);",

            # 9. Индекс для Employee (поиск по имени)
            "CREATE INDEX IF NOT EXISTS idx_employee_name ON employee
(last_name, first_name);",

            # 10. Индекс для Store по классу
            "CREATE INDEX IF NOT EXISTS idx_store_class ON store
(store_class_id);"
        ]

        with engine.connect() as conn:
            for sql in indexes_sql:
                try:
                    conn.execute(text(sql))
                    conn.commit()
                except Exception as e:
                    print(f"Ошибка при создании индекса: {e}")


    def drop_indexes(engine):
        print("Удаление индексов...")

        drop_sql = [
            "DROP INDEX IF EXISTS idx_product_name_search;",
            "DROP INDEX IF EXISTS idx_department_store_id_fk;",
            "DROP INDEX IF EXISTS idx_department_product_count_sort;",
            "DROP INDEX IF EXISTS idx_warehouse_product_count_sort;",
            "DROP INDEX IF EXISTS idx_warehouse_priority_sort;",
            "DROP INDEX IF EXISTS idx_product_price_sort;",
            "DROP INDEX IF EXISTS idx_warehouse_product_price_sort;",
            "DROP INDEX IF EXISTS idx_department_product_join;",
            "DROP INDEX IF EXISTS idx_warehouse_product_join;",
            "DROP INDEX IF EXISTS idx_product_price_join;",
            "DROP INDEX IF EXISTS idx_employee_name;",
            "DROP INDEX IF EXISTS idx_store_class;"
        ]

        with engine.connect() as conn:
            for sql in drop_sql:
```

```
                try:
                    conn.execute(text(sql))
                    conn.commit()
                except Exception as e:
                    print(f"Ошибка при удалении индекса: {e}")


    def populate_test_data(session):
        # Очистка существующих данных
        session.execute(text(truncate_sql))

        # Классы магазинов
        store_classes = [
            StoreClass(name='Элитный', description='Магазины премиум-класса с широким ассортиментом'),
            StoreClass(name='Стандарт', description='Магазины среднего ценового сегмента'),
            StoreClass(name='Эконом', description='Бюджетные магазины'),
            StoreClass(name='Специализированный', description='Магазины узкой направленности'),
            StoreClass(name='Гипермаркет', description='Крупные торговые центры')
        ]
        session.add_all(store_classes)
        session.flush()

        # Торговые базы
        trading_bases = [
            TradingBase(name='Центральная база "Северная"', description='Крупнейший распределительный центр в северном регионе'),
            TradingBase(name='Южный распределительный центр', description='Современный логистический комплекс южного направления'),
            TradingBase(name='Западный складской комплекс', description='Складские помещения с системой климат-контроля'),
            TradingBase(name='Восточная база снабжения', description='База снабжения для розничных сетей восточного региона'),
            TradingBase(name='Центральный логистический центр', description='Основной хаб для федеральных сетей')
        ]
        session.add_all(trading_bases)
        session.flush()

        # Сотрудники
        employees = [
            Employee(first_name='Иван', last_name='Петров'),
            Employee(first_name='Мария', last_name='Сидорова'),
            Employee(first_name='Алексей', last_name='Козлов'),
            Employee(first_name='Ольга', last_name='Николаева'),
            Employee(first_name='Сергей', last_name='Васильев'),
            Employee(first_name='Елена', last_name='Федорова'),
            Employee(first_name='Дмитрий', last_name='Орлов'),
            Employee(first_name='Анна', last_name='Морозова'),
            Employee(first_name='Павел', last_name='Семенов'),
            Employee(first_name='Ирина', last_name='Волкова')
        ]
        session.add_all(employees)
        session.flush()

        # Магазины
        stores = [
            Store(name='Супермаркет "Восток"', description='Крупный супермаркет в центре города',
```

```
                    store_class_id=store_classes[1].store_class_id,
director_id=employees[0].employee_id),
            Store(name='Гипермаркет "Мега"', description='Торговый центр с
полным ассортиментом',
                    store_class_id=store_classes[4].store_class_id,
director_id=employees[3].employee_id),
            Store(name='Магазин "Эконом"', description='Бюджетный магазин для
ежедневных покупок',
                    store_class_id=store_classes[2].store_class_id,
director_id=employees[5].employee_id),
            Store(name='Премиум маркет "Люкс"', description='Магазин премиум-
класса',
                    store_class_id=store_classes[0].store_class_id,
director_id=employees[8].employee_id),
            Store(name='Спецмагазин                                    "Техника"',
description='Специализированный магазин электроники',
                    store_class_id=store_classes[3].store_class_id,
director_id=employees[3].employee_id)
        ]
        session.add_all(stores)
        session.flush()

        # Отделы
        departments = [
            Department(store_id=stores[0].store_id,
manager_id=employees[1].employee_id, name='Молочный отдел'),
            Department(store_id=stores[0].store_id,
manager_id=employees[2].employee_id, name='Хлебный отдел'),
            Department(store_id=stores[1].store_id,
manager_id=employees[4].employee_id, name='Овощной отдел'),
            Department(store_id=stores[1].store_id,
manager_id=employees[6].employee_id, name='Мясной отдел'),
            Department(store_id=stores[2].store_id,
manager_id=employees[7].employee_id, name='Бакалея'),
            Department(store_id=stores[3].store_id,
manager_id=employees[9].employee_id, name='Гастрономия'),
            Department(store_id=stores[3].store_id,
manager_id=employees[1].employee_id, name='Кондитерский отдел'),
            Department(store_id=stores[4].store_id,
manager_id=employees[2].employee_id, name='Бытовая техника'),
            Department(store_id=stores[4].store_id,
manager_id=employees[4].employee_id, name='Электроника'),
            Department(store_id=stores[0].store_id,
manager_id=employees[6].employee_id, name='Напитки')
        ]
        session.add_all(departments)
        session.flush()

        # Товары
        products = [
            Product(name='Молоко 2,5%', sort='Пастеризованное'),
            Product(name='Хлеб Бородинский', sort='Ржаной'),
            Product(name='Картофель', sort='Отборный'),
            Product(name='Курица охлажденная', sort='Бройлер'),
            Product(name='Рис круглый', sort='Высший сорт'),
            Product(name='Кофе молотый', sort='Арабика'),
            Product(name='Шоколад горький', sort='Премиум'),
            Product(name='Чай черный', sort='Цейлон'),
            Product(name='Сок яблочный', sort='Осветленный'),
            Product(name='Телевизор LED', sort='Smart TV'),
            Product(name='Смартфон', sort='Флагман'),
            Product(name='Йогурт натуральный', sort='Без добавок'),
            Product(name='Сыр Российский', sort='Полутвердый'),
```

```
    Product(name='Колбаса докторская', sort='Вареная'),
    Product(name='Печенье овсяное', sort='С шоколадом')
]
session.add_all(products)
session.flush()

# Товары в отделах
department_products = [
    (1, 1, 0), (1, 12, 30), (1, 13, 25),
    (2, 2, 40), (2, 15, 45),
    (3, 3, 100), (3, 5, 60),
    (4, 4, 35), (4, 14, 28),
    (5, 5, 55), (5, 6, 20), (5, 7, 35),
    (6, 7, 15), (6, 8, 25), (6, 13, 18),
    (7, 7, 30), (7, 15, 40),
    (8, 10, 8), (9, 11, 12),
    (10, 1, 25), (10, 9, 35)
]

for dept_id, art, cnt in department_products:
    session.add(DepartmentProduct(
        department_id=departments[dept_id-1].department_id,
        article=products[art-1].article,
        count=cnt
    ))

# Товары на складах баз
warehouse_products = [
    (1, 1, 200, 85.50), (1, 2, 150, 45.00), (1, 3, 500, 35.00),
    (2, 4, 180, 320.00), (2, 5, 300, 95.00), (2, 6, 120, 450.00),
    (3, 7, 90, 180.00), (3, 8, 200, 120.00), (3, 9, 150, 110.00),
    (4, 10, 25, 25000.00), (4, 11, 40, 45000.00),
    (5, 12, 100, 65.00), (5, 13, 80, 580.00), (5, 14, 60, 420.00),
    (1, 15, 120, 85.00), (2, 1, 180, 82.00)
]

for base_id, art, cnt, price in warehouse_products:
    session.add(WarehouseProduct(
        trading_base_id=trading_bases[base_id-1].trading_base_id,
        article=products[art-1].article,
        count=cnt,
        price=price
    ))

# Цены товаров по классам магазинов
product_prices = [
    (1, 1, 120.00), (1, 2, 65.00), (1, 3, 50.00),
    (2, 1, 95.00), (2, 2, 48.00), (2, 3, 38.00),
    (3, 1, 80.00), (3, 2, 40.00), (3, 3, 30.00),
    (1, 7, 250.00), (2, 7, 200.00), (3, 7, 150.00),
    (4, 10, 28000.00), (5, 10, 27000.00)
]

for class_id, art, price in product_prices:
    session.add(ProductPrice(
        store_class_id=store_classes[class_id-1].store_class_id,
        article=products[art-1].article,
        price=price
    ))

# Приоритеты поставок
warehouse_priorities = [
    (1, 1, 1, 1), (1, 2, 1, 2), (1, 3, 1, 1),
```

```
            (2, 1, 1, 2), (2, 2, 1, 3), (2, 3, 1, 2),
            (3, 1, 1, 1), (3, 2, 1, 1), (3, 3, 1, 1),
            (10, 4, 4, 1), (10, 5, 4, 1), (11, 5, 4, 1),
            (7, 4, 3, 2), (7, 1, 3, 3),
            (1, 1, 2, 2), (2, 1, 2, 3)
        ]

        for art, store_id, base_id, priority in warehouse_priorities:
            session.add(WarehousePriority(
                article=products[art-1].article,
                store_id=stores[store_id-1].store_id,
                trading_base_id=trading_bases[base_id-1].trading_base_id,
                priority=priority
            ))

        session.commit()
        print("Тестовые данные успешно добавлены!")
```

queries.py:

```
    from sqlalchemy import func, and_, or_, not_, case, select
    from sqlalchemy.orm import aliased
    from models import WarehousePriority, ProductPrice, WarehouseProduct,
DepartmentProduct, Product, Department, Store, Employee, TradingBase, StoreClass
    import time

    def execute_queries(session, show_output=True):
        print("=" * 80)
        print("ЗАПРОСЫ К БАЗЕ ДАННЫХ УНИВЕРСИТЕТСКОГО МАГАЗИНА")
        print("=" * 80)

        # Список всех запросов для выполнения
        queries = [
            query_1_1_store_products,
            query_1_2_base_products,
            query_2_1_orderable_products,
            query_2_2_extended_orderable_products,
            query_3_department_products,
            query_4_department_managers,
            query_5_department_values,
            query_6_product_search
        ]

        # Выполнение всех запросов
        for query_func in queries:
            query_func(session, show_output=show_output)

    def query_1_1_store_products(session, store_id=1, show_output=True):
        """1.1 Какие товары имеются в магазине?"""
        start_time = time.time()

        if show_output:
            print(f"\n1.1 ТОВАРЫ В МАГАЗИНЕ ID={store_id}:")
            print("-" * 60)

        store_products = (session.query(
            Store.name.label('store_name'),
            Department.name.label('department_name'),
            Product.article,
            Product.name.label('product_name'),
            Product.sort.label('product_sort'),
            DepartmentProduct.count.label('quantity'),
            ProductPrice.price.label('current_price'),
```

```
            (DepartmentProduct.count                                                    *
ProductPrice.price).label('total_value')
        )
        .select_from(Store)
        .join(Department, Store.store_id == Department.store_id)
        .join(DepartmentProduct,          Department.department_id          ==
DepartmentProduct.department_id)
        .join(Product, DepartmentProduct.article == Product.article)
        .join(ProductPrice, and_(
            Product.article == ProductPrice.article,
            Store.store_class_id == ProductPrice.store_class_id
        ))
        .filter(Store.store_id == store_id)
        .order_by(Department.name, Product.name)
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time

        if show_output:
            for product in store_products:
                print(f"Отдел:       {product.department_name:20}       Товар:
{product.product_name:25} "
                      f"Кол-во:           {product.quantity:3}           Цена:
{float(product.current_price):8.2f} "
                      f"Сумма: {float(product.total_value):10.2f}")

        print(f"Запрос 1.1 выполнен за {execution_time:.4f} секунд, обработано
{len(store_products)} записей")
        return store_products, execution_time

    def query_1_2_base_products(session, base_id=2, show_output=True):
        """1.2 Какие товары имеются на базе?"""
        start_time = time.time()

        if show_output:
            print(f"\n1.2 ТОВАРЫ НА ТОРГОВОЙ БАЗЕ ID={base_id}:")
            print("-" * 60)

        base_products = (session.query(
            TradingBase.name.label('trading_base_name'),
            Product.article,
            Product.name.label('product_name'),
            Product.sort.label('product_sort'),
            WarehouseProduct.count.label('available_quantity'),
            WarehouseProduct.price.label('base_price'),
            (WarehouseProduct.count                                                    *
WarehouseProduct.price).label('total_value')
        )
        .select_from(TradingBase)
        .join(WarehouseProduct,         TradingBase.trading_base_id         ==
WarehouseProduct.trading_base_id)
        .join(Product, WarehouseProduct.article == Product.article)
        .filter(TradingBase.trading_base_id == base_id)
        .filter(WarehouseProduct.count > 0)
        .order_by(Product.name)
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time
```

```
        if show_output:
            for product in base_products:
                print(f"Товар:        {product.product_name:25}        Кол-во:
{product.available_quantity:4} "
                      f"Цена:      {float(product.base_price):8.2f}      Сумма:
{float(product.total_value):12.2f}")

        print(f"Запрос 1.2 выполнен за {execution_time:.4f} секунд, обработано
{len(base_products)} записей")
        return base_products, execution_time

    def query_2_1_orderable_products(session, store_id=1, show_output=True):
        """2. Какие отсутствующие товары может заказать магазин на базе?"""
        start_time = time.time()

        if show_output:
            print(f"\n2.   ОТСУТСТВУЮЩИЕ   ТОВАРЫ   ДЛЯ   ЗАКАЗА   МАГАЗИНОМ
ID={store_id}:")
            print("-" * 60)

        # Подзапрос для товаров с нулевым количеством в магазине
        zero_products_subquery = select(DepartmentProduct.article).\
            join(Department,         DepartmentProduct.department_id         ==
Department.department_id).\
            where(and_(
                Department.store_id == store_id,
                DepartmentProduct.count == 0
            )).\
            scalar_subquery()

        orderable_products = (session.query(
            Store.name.label('store_name'),
            Product.name.label('product_name'),
            TradingBase.name.label('trading_base'),
            WarehouseProduct.count.label('available_quantity'),
            WarehouseProduct.price.label('base_price'),
            WarehousePriority.priority
        )
        .select_from(Product)
        .join(WarehouseProduct, Product.article == WarehouseProduct.article)
        .join(TradingBase,         WarehouseProduct.trading_base_id         ==
TradingBase.trading_base_id)
        .join(Store, Store.store_id == store_id)
        .join(WarehousePriority, and_(
            WarehousePriority.store_id == Store.store_id,
            WarehousePriority.article == Product.article,
            WarehousePriority.trading_base_id == TradingBase.trading_base_id
        ))
        .filter(Product.article.in_(zero_products_subquery))
        .filter(WarehouseProduct.count >= 1)
        .order_by(Product.name, WarehousePriority.priority)
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time

        if show_output:
            for product in orderable_products:
                print(f"Товар:         {product.product_name:25}         База:
{product.trading_base:30} "
                      f"Доступно:     {product.available_quantity:3}     Цена:
{float(product.base_price):8.2f} "
```

```
                    f"Приоритет: {product.priority}")

        print(f"Запрос 2 выполнен за {execution_time:.4f} секунд, обработано
{len(orderable_products)} записей")
        return orderable_products, execution_time

    def query_2_2_extended_orderable_products(session, store_id=1,
show_output=True):
        """2.2 Какие отсутствующие товары может заказать магазин на базе
(включая новые товары)?"""
        start_time = time.time()

        if show_output:
            print(f"\n2.2 ОТСУТСТВУЮЩИЕ ТОВАРЫ ДЛЯ ЗАКАЗА МАГАЗИНОМ
ID={store_id} (включая новые товары):")
            print("-" * 60)

        # Подзапрос для товаров, которые есть в магазине
        store_products_subquery = select(DepartmentProduct.article).\
            join(Department, DepartmentProduct.department_id ==
Department.department_id).\
            where(Department.store_id == store_id)

        # Подзапрос для товаров с нулевым количеством
        zero_count_subquery = select(DepartmentProduct.article).\
            join(Department, DepartmentProduct.department_id ==
Department.department_id).\
            where(and_(
                Department.store_id == store_id,
                DepartmentProduct.count == 0
            ))

        extended_orderable_products = (session.query(
            Store.name.label('store_name'),
            Product.name.label('product_name'),
            TradingBase.name.label('trading_base'),
            WarehouseProduct.count.label('available_quantity'),
            WarehouseProduct.price.label('base_price'),
            WarehousePriority.priority,
            case(
                (Product.article.in_(zero_count_subquery), 'ЗАКОНЧИЛСЯ'),
                (not_(Product.article.in_(store_products_subquery)), 'НОВЫЙ
ТОВАР'),
                else_='ДРУГОЙ'
            ).label('status')
        )
        .select_from(Product)
        .join(WarehouseProduct, Product.article == WarehouseProduct.article)
        .join(TradingBase, WarehouseProduct.trading_base_id ==
TradingBase.trading_base_id)
        .join(Store, Store.store_id == store_id)
        .join(WarehousePriority, and_(
            WarehousePriority.store_id == Store.store_id,
            WarehousePriority.article == Product.article,
            WarehousePriority.trading_base_id == TradingBase.trading_base_id
        ))
        .filter(or_(
            Product.article.in_(zero_count_subquery),
            not_(Product.article.in_(store_products_subquery))
        ))
        .filter(WarehouseProduct.count >= 0)
        .order_by(Product.name, WarehousePriority.priority)
        .limit(100)
```

```
            .all())

            end_time = time.time()
            execution_time = end_time - start_time

            if show_output:
                if extended_orderable_products:
                    for product in extended_orderable_products:
                        status_text = "ЗАКОНЧИЛСЯ" if product.status ==
'ЗАКОНЧИЛСЯ' else "НОВЫЙ ТОВАР (может быть добавлен)"
                        print(f"Товар: {product.product_name:25} Статус:
{status_text:25} "
                              f"База: {product.trading_base:25} Приоритет:
{product.priority}")
                else:
                    print("Нет товаров для заказа")

            print(f"Запрос 2.2 выполнен за {execution_time:.4f} секунд, обработано
{len(extended_orderable_products)} записей")
            return extended_orderable_products, execution_time

        def query_3_department_products(session, department_id=1, store_id=1,
show_output=True):
            """3. Какие товары, и в каком количестве имеются в отделе магазина?"""
            start_time = time.time()

            if show_output:
                print(f"\n3. ТОВАРЫ В ОТДЕЛЕ ID={department_id} МАГАЗИНА
ID={store_id}:")
                print("-" * 60)

            department_products = (session.query(
                Store.name.label('store_name'),
                Department.name.label('department_name'),
                Product.name.label('product_name'),
                DepartmentProduct.count.label('quantity')
            )
            .select_from(Store)
            .join(Department, Store.store_id == Department.store_id)
            .join(DepartmentProduct, Department.department_id ==
DepartmentProduct.department_id)
            .join(Product, DepartmentProduct.article == Product.article)
            .filter(Department.department_id == department_id)
            .order_by(Product.name)
            .limit(100)
            .all())

            end_time = time.time()
            execution_time = end_time - start_time

            if show_output:
                for product in department_products:
                    print(f"Магазин: {product.store_name:25} Отдел:
{product.department_name:20} "
                          f"Товар: {product.product_name:25} Кол-во:
{product.quantity}")

            print(f"Запрос 3 выполнен за {execution_time:.4f} секунд, обработано
{len(department_products)} записей")
            return department_products, execution_time

        def query_4_department_managers(session, store_id=1, show_output=True):
            """4. Список заведующих отделами магазина"""
```

```
        start_time = time.time()

        if show_output:
            print(f"\n4. ЗАВЕДУЮЩИЕ ОТДЕЛАМИ МАГАЗИНА ID={store_id}:")
            print("-" * 60)

        department_managers = (session.query(
            Store.name.label('store_name'),
            Department.name.label('department_name'),
            func.concat(Employee.first_name,                ',
Employee.last_name).label('manager_name')
        )
        .select_from(Store)
        .join(Department, Store.store_id == Department.store_id)
        .join(Employee, Department.manager_id == Employee.employee_id)
        .filter(Store.store_id == store_id)
        .order_by(Department.name)
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time

        if show_output:
            for manager in department_managers:
                print(f"Отдел:    {manager.department_name:20}    Заведующий:
{manager.manager_name}")

        print(f"Запрос 4 выполнен за {execution_time:.4f} секунд, обработано
{len(department_managers)} записей")
        return department_managers, execution_time

    def query_5_department_values(session, show_output=True):
        """5. Суммарная стоимость товара в каждом отделе"""
        start_time = time.time()

        if show_output:
            print(f"\n5. СУММАРНАЯ СТОИМОСТЬ ТОВАРОВ ПО ОТДЕЛАМ:")
            print("-" * 60)

        department_values = (session.query(
            Store.name.label('store_name'),
            Department.name.label('department_name'),
            func.sum(DepartmentProduct.count                                *
ProductPrice.price).label('total_value')
        )
        .select_from(Store)
        .join(Department, Store.store_id == Department.store_id)
        .join(DepartmentProduct,          Department.department_id          ==
DepartmentProduct.department_id)
        .join(ProductPrice, and_(
            DepartmentProduct.article == ProductPrice.article,
            Store.store_class_id == ProductPrice.store_class_id
        ))
        .group_by(Store.name, Department.name)
        .order_by(func.sum(DepartmentProduct.count                          *
ProductPrice.price).desc())
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time
```

```
        if show_output:
            for dept in department_values:
                print(f"Магазин:          {dept.store_name:25}          Отдел:
{dept.department_name:20} "
                      f"Сумма: {float(dept.total_value):12.2f}")

        print(f"Запрос 5 выполнен за {execution_time:.4f} секунд, обработано
{len(department_values)} записей")
        return department_values, execution_time

    def         query_6_product_search(session,         product_name='Молоко',
show_output=True):
        """6. На каких базах, и в каких количествах есть товар нужного
наименования?"""
        start_time = time.time()

        if show_output:
            print(f"\n6. ПОИСК ТОВАРА '{product_name}' ПО БАЗАМ:")
            print("-" * 60)

        search_products = (session.query(
            Product.name.label('product_name'),
            TradingBase.name.label('trading_base'),
            TradingBase.description.label('base_description'),
            WarehouseProduct.count.label('available_quantity'),
            WarehouseProduct.price.label('price_per_unit')
        )
        .select_from(Product)
        .join(WarehouseProduct, Product.article == WarehouseProduct.article)
        .join(TradingBase,         WarehouseProduct.trading_base_id         ==
TradingBase.trading_base_id)
        .filter(Product.name.ilike(f'%{product_name}%'))
        .order_by(WarehouseProduct.count.desc())
        .limit(100)
        .all())

        end_time = time.time()
        execution_time = end_time - start_time

        if show_output:
            for product in search_products:
                print(f"Товар:         {product.product_name:25}         База:
{product.trading_base:35} "
                      f"Кол-во:      {product.available_quantity:4}      Цена:
{float(product.price_per_unit):8.2f}")

        print(f"Запрос 6 выполнен за {execution_time:.4f} секунд, обработано
{len(search_products)} записей")
        return search_products, execution_time
```