Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Уральский федеральный университет

имени первого Президента России Б.Н.Ельцина»

Институт радиоэлектроники и информационных технологий

Отчёт

по лабораторной работе №1

«Основы работы с Docker и PostgreSQL»

Выполнил: студент гр. РИМ-150950

Коренева Екатерина Александровна

Екатеринбург

2025 г.

**Цель работы**

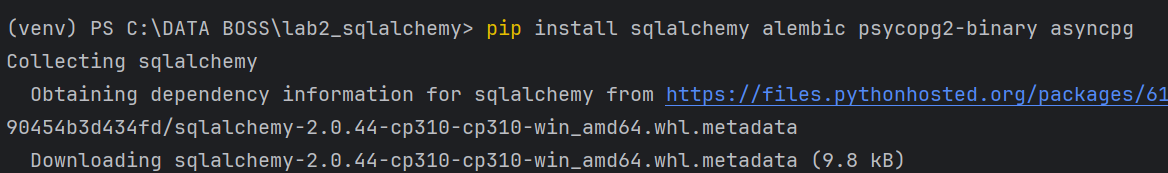
Освоить принципы работы с библиотеками SQLAlchemy и Alembic для создания и управления реляционными базами данных на Python, изучить механизмы миграции базы данных.

Выполняемые задачи:

1. Установка необходимых библиотек
2. Создание ORM-моделей
3. Выполнить запросы с использованием selectinload()
4. Расширение структуры и добавление новых таблиц

## Инициализация БД

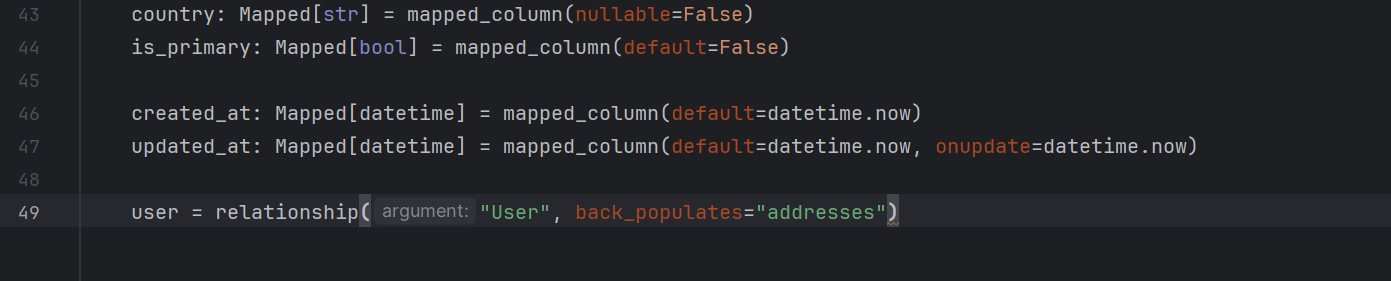
1. Установка библиотек

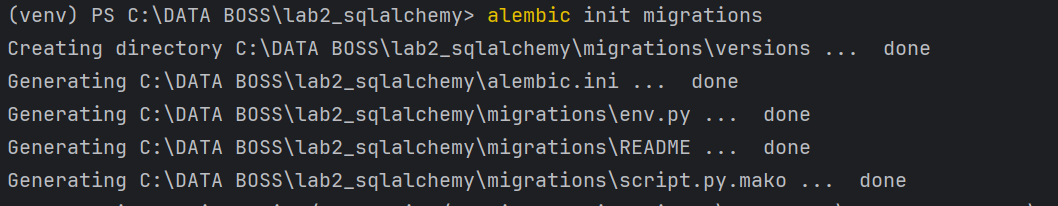


Создаем файл models.py в папке проекта и в нем Создаем ОРМ для пользователя и адреса

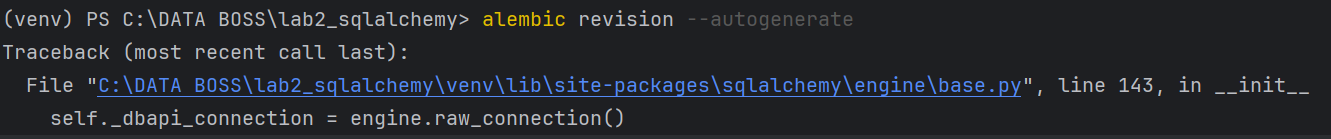




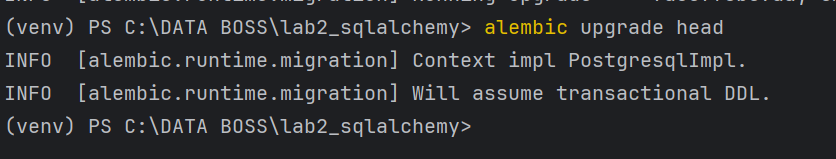


1. Инициализируем миграции
2. 
3. Меняем в alembic.ini строку подключения к БД: sqlalchemy.url = postgresql://postgres:postgres@192.168.99.100:5432/postgres
4. В env.py подтсавляем метаданные для миграции import sys

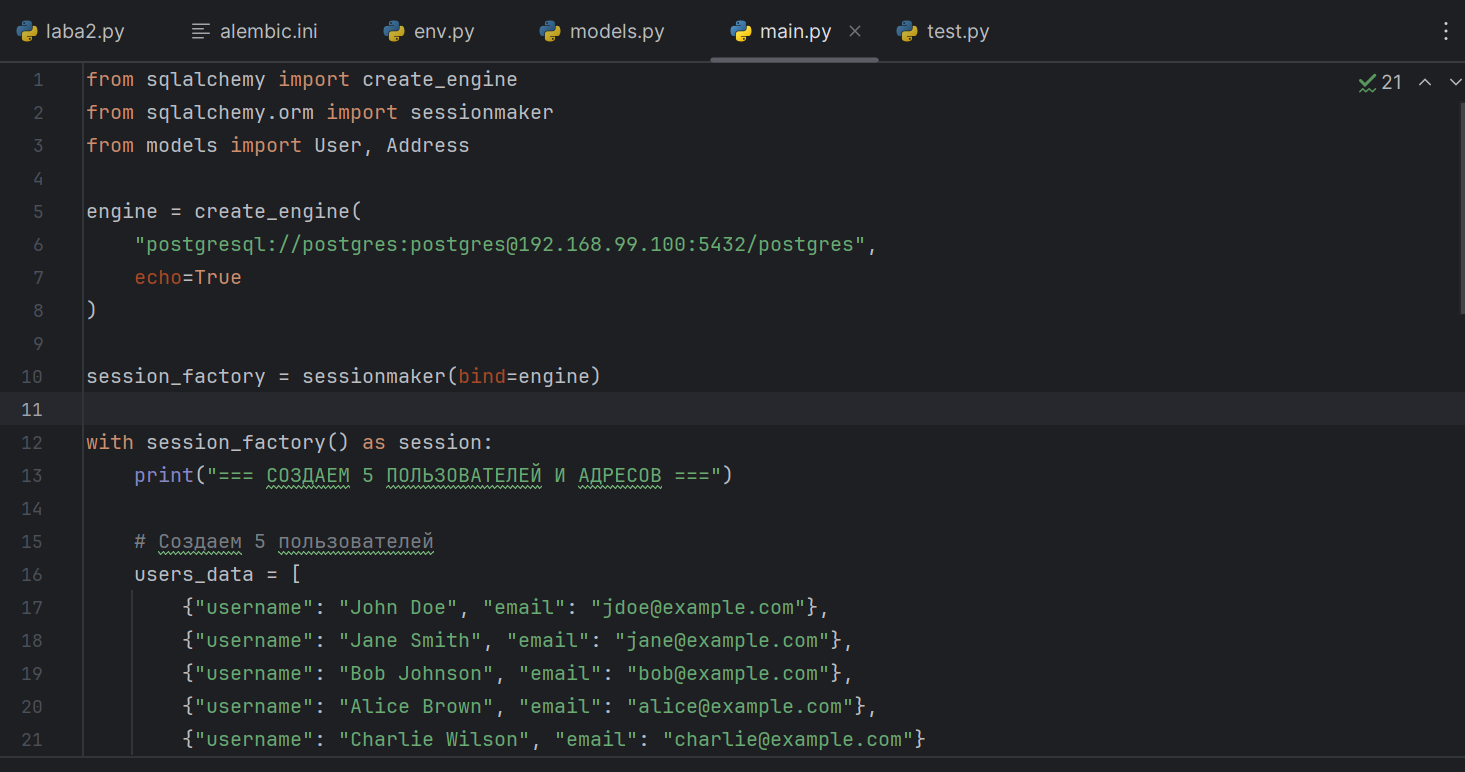
sys.path.append("C:/DATA BOSS/lab2\_sqlalchemy")

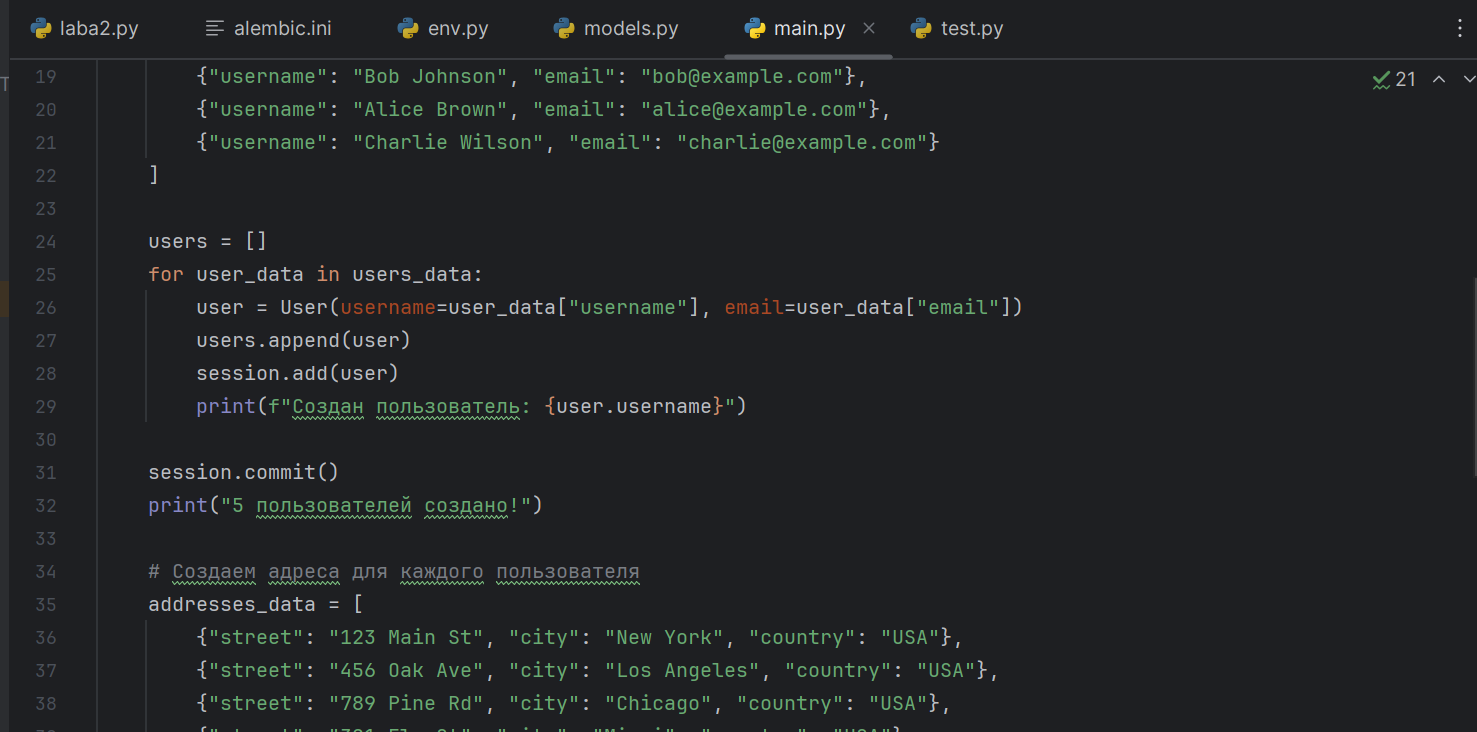
from models import Base  
  
 Создаем миграцию   
****

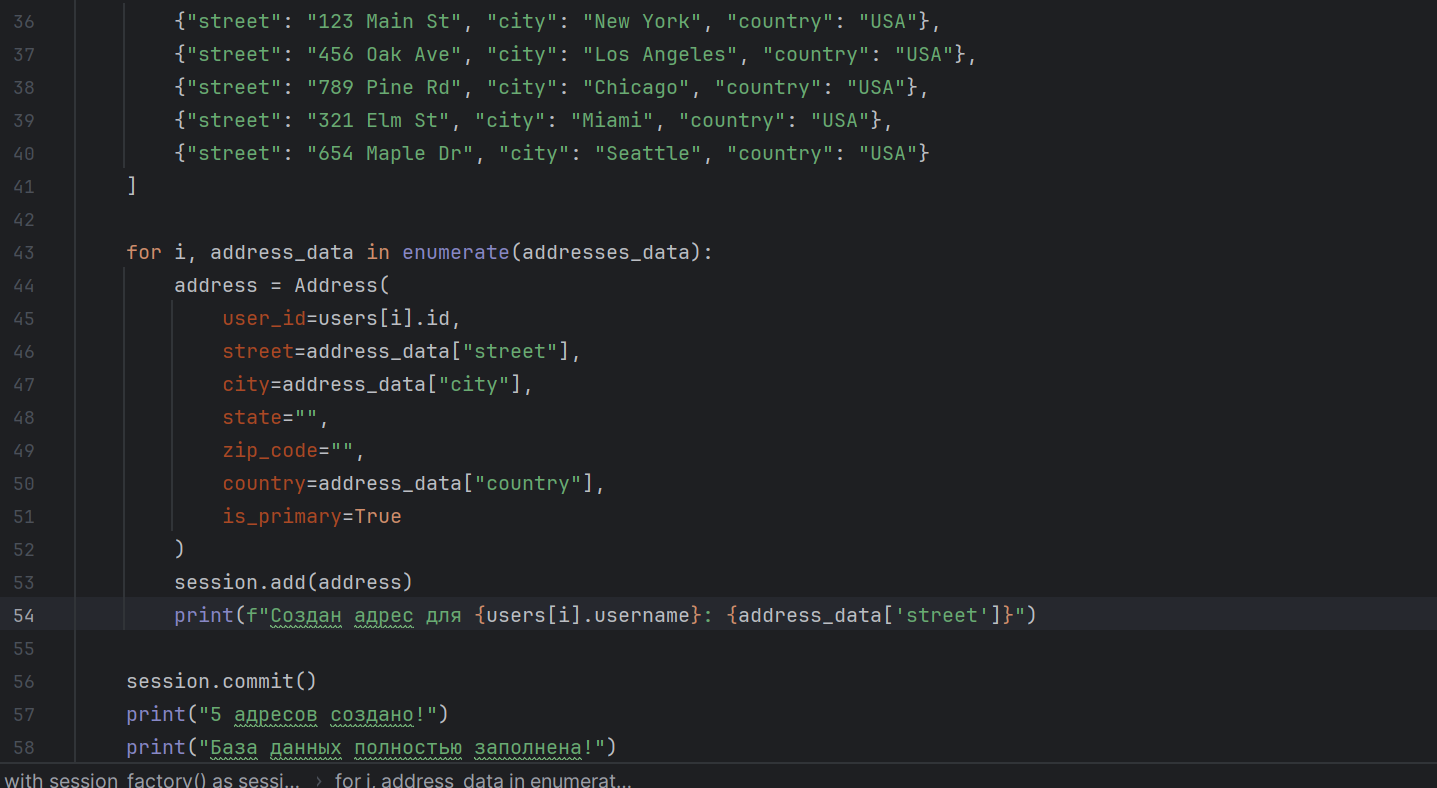
1. Применяем к БД последнюю миграцию

****

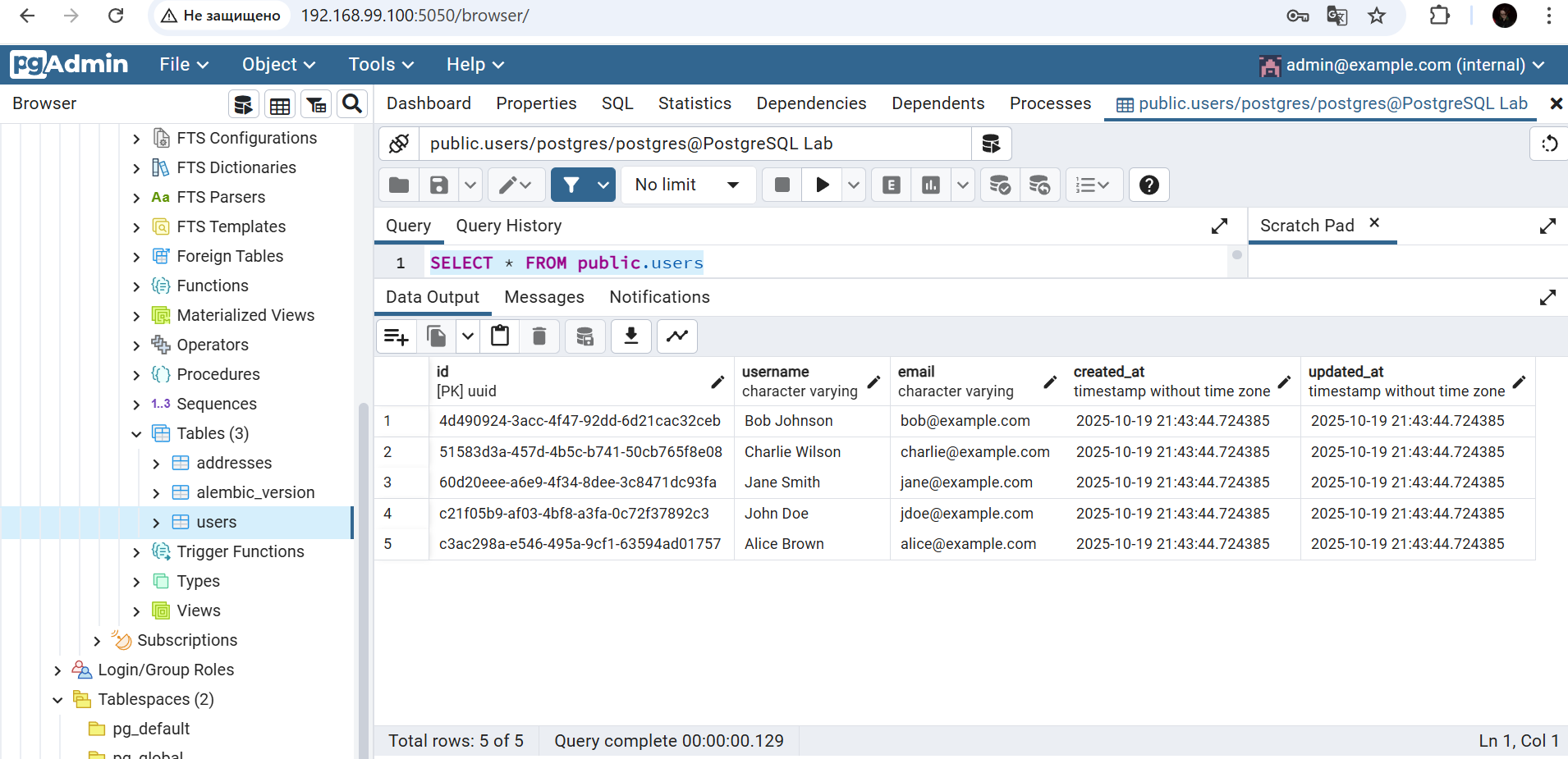
1. Наполняем БД данными, пользователь и электронный адрес

****

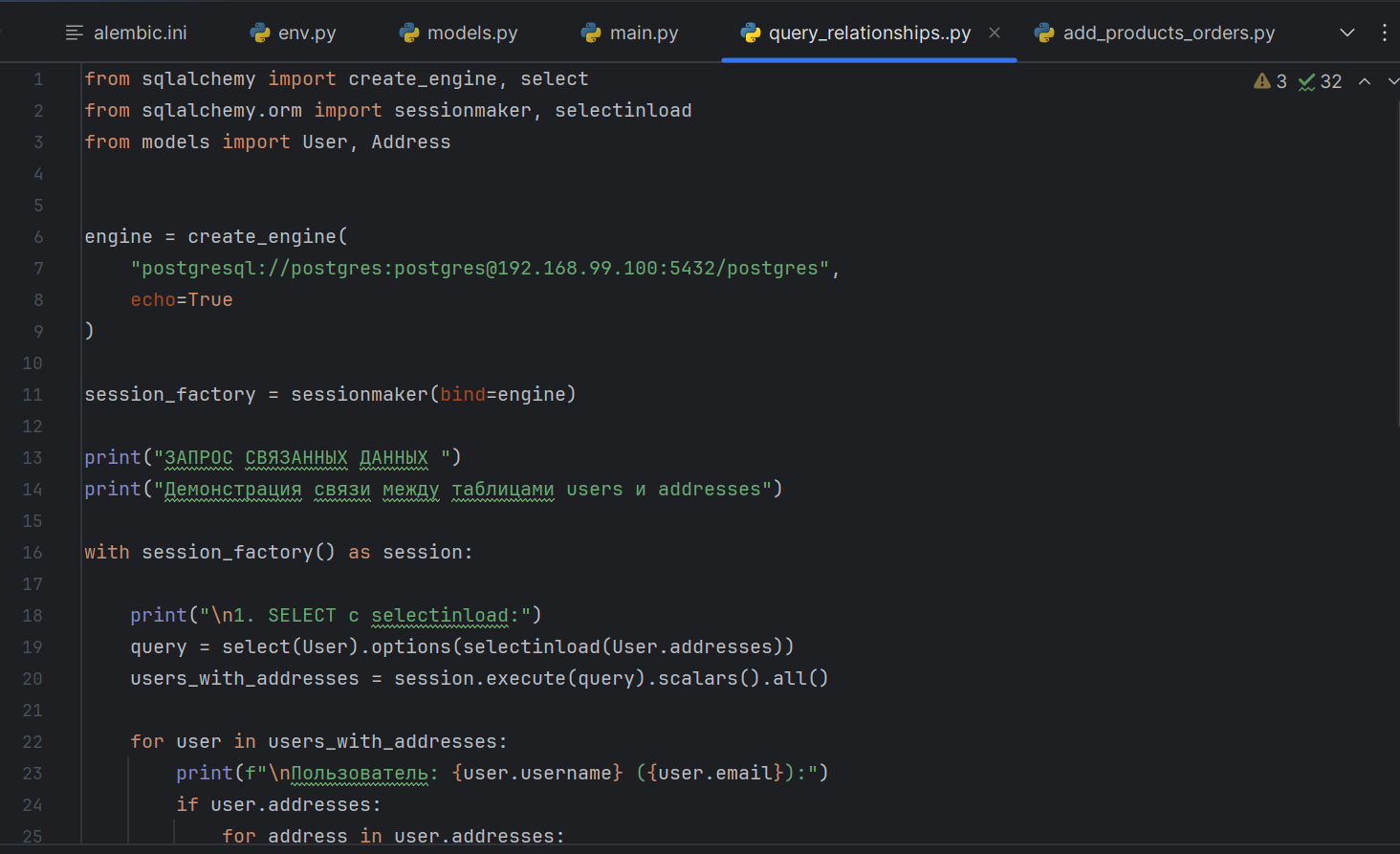
****

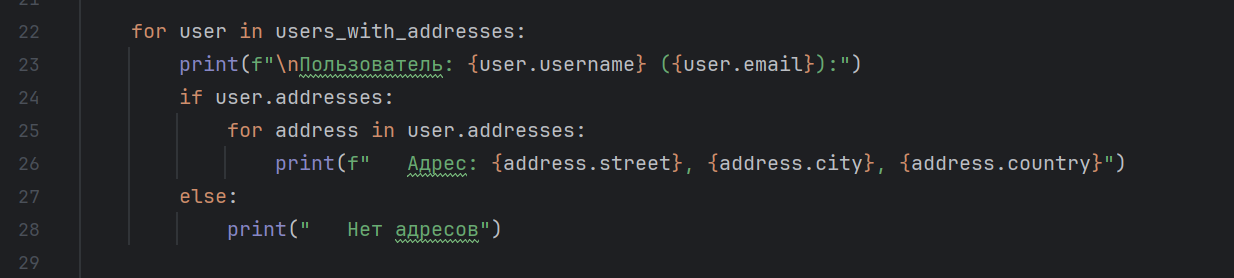
****

**Результат выполнения кода:**

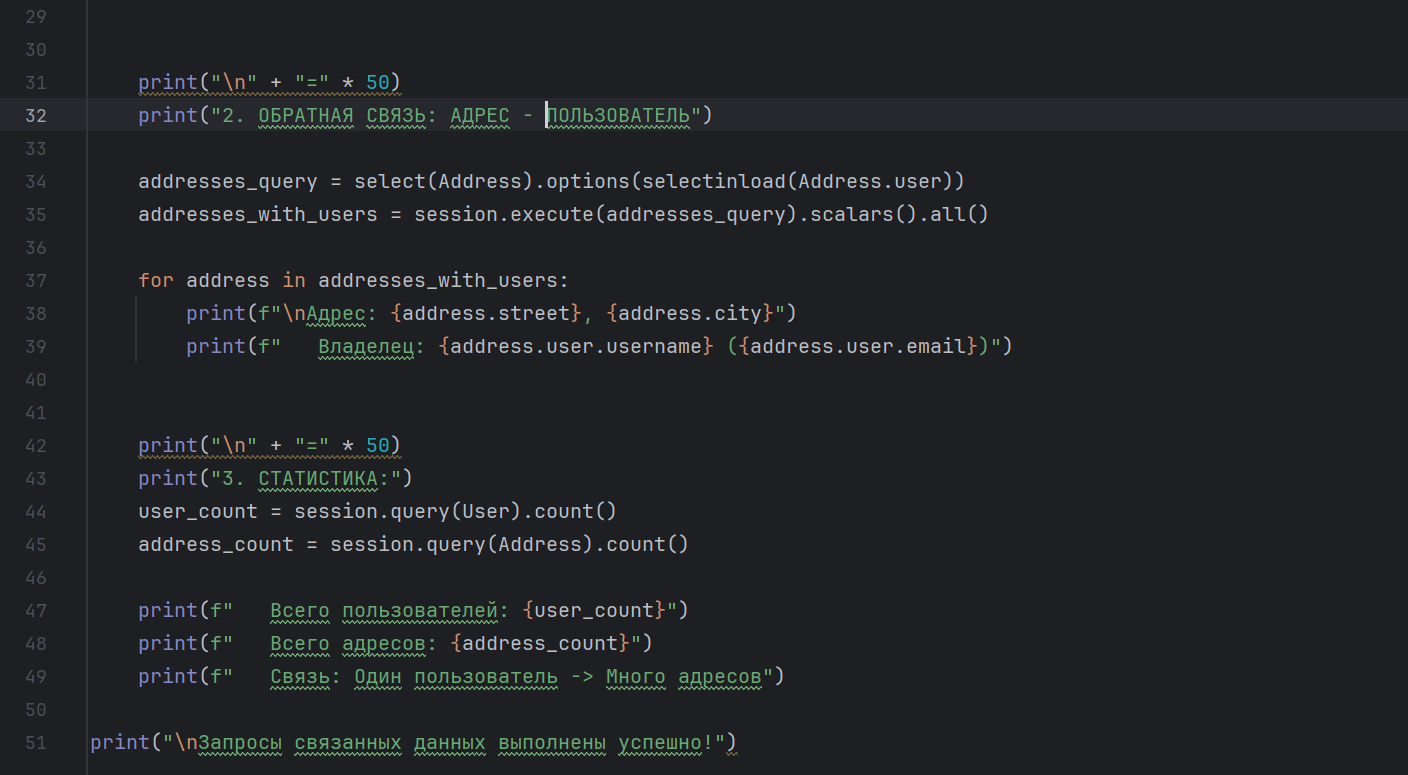


1. Запрос связанных данных  
   в коде выполнено: запрос пользователей с их адресами:





Запрос адресов с информацией о владельцах:



## Последующие работы с БД и миграции

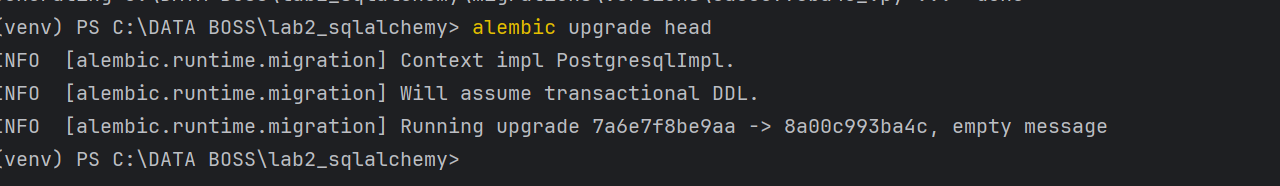
## Добавление поля description (строка 20)

## 

## 

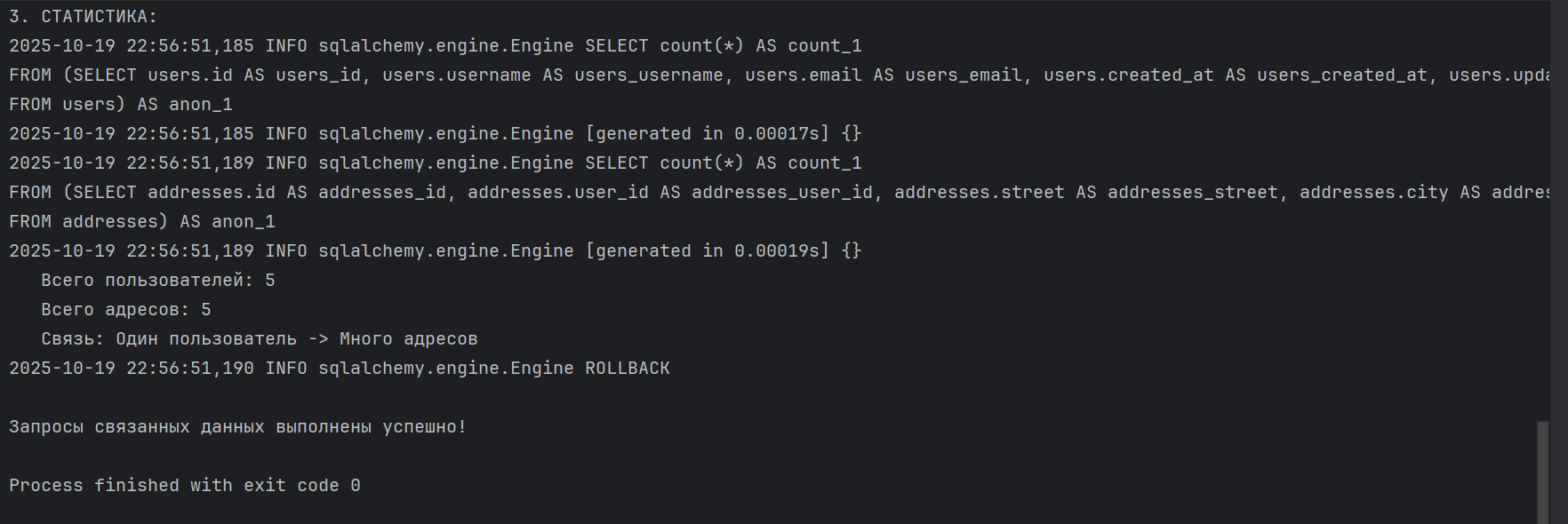
## 

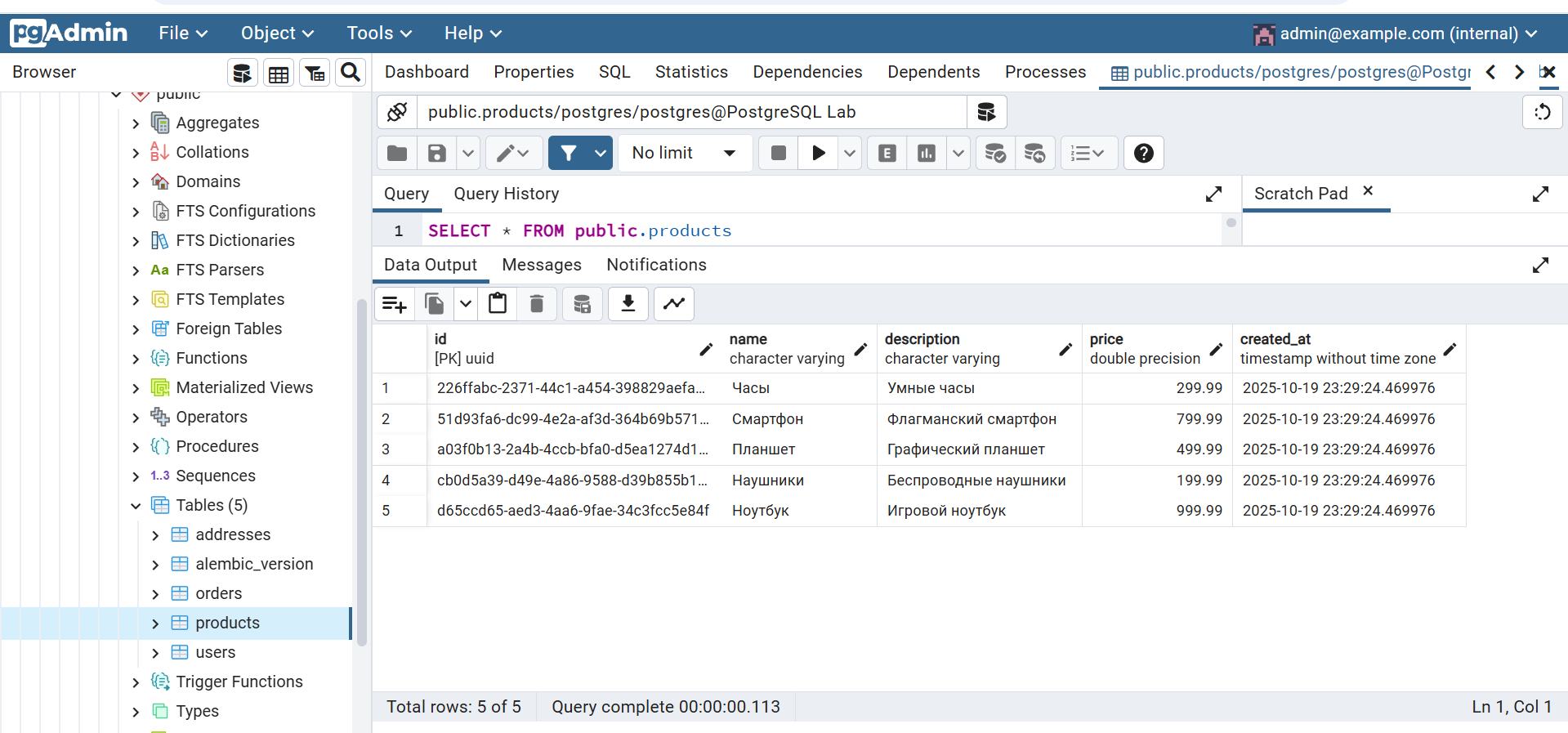
## 

****

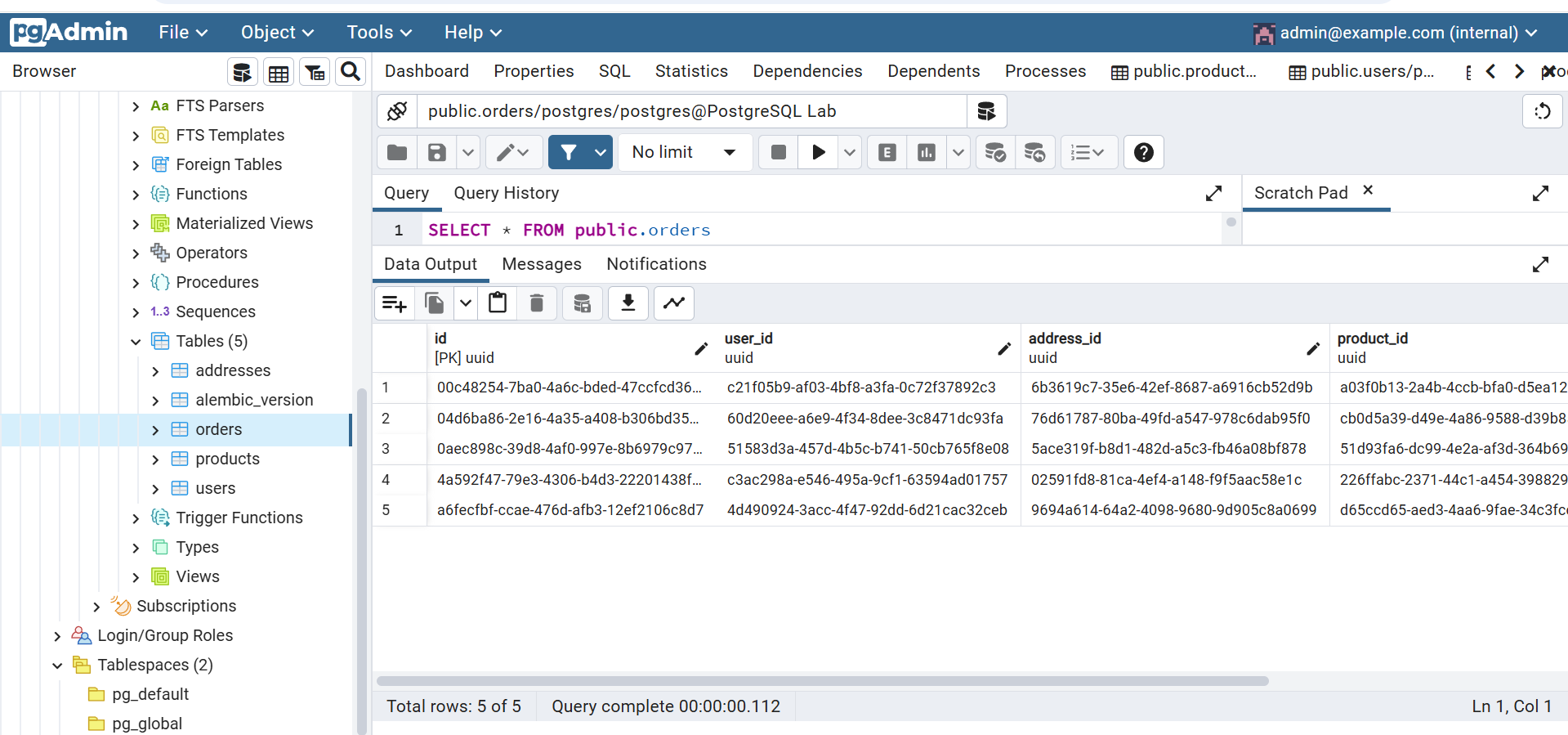
**Добавляем новые продукты**

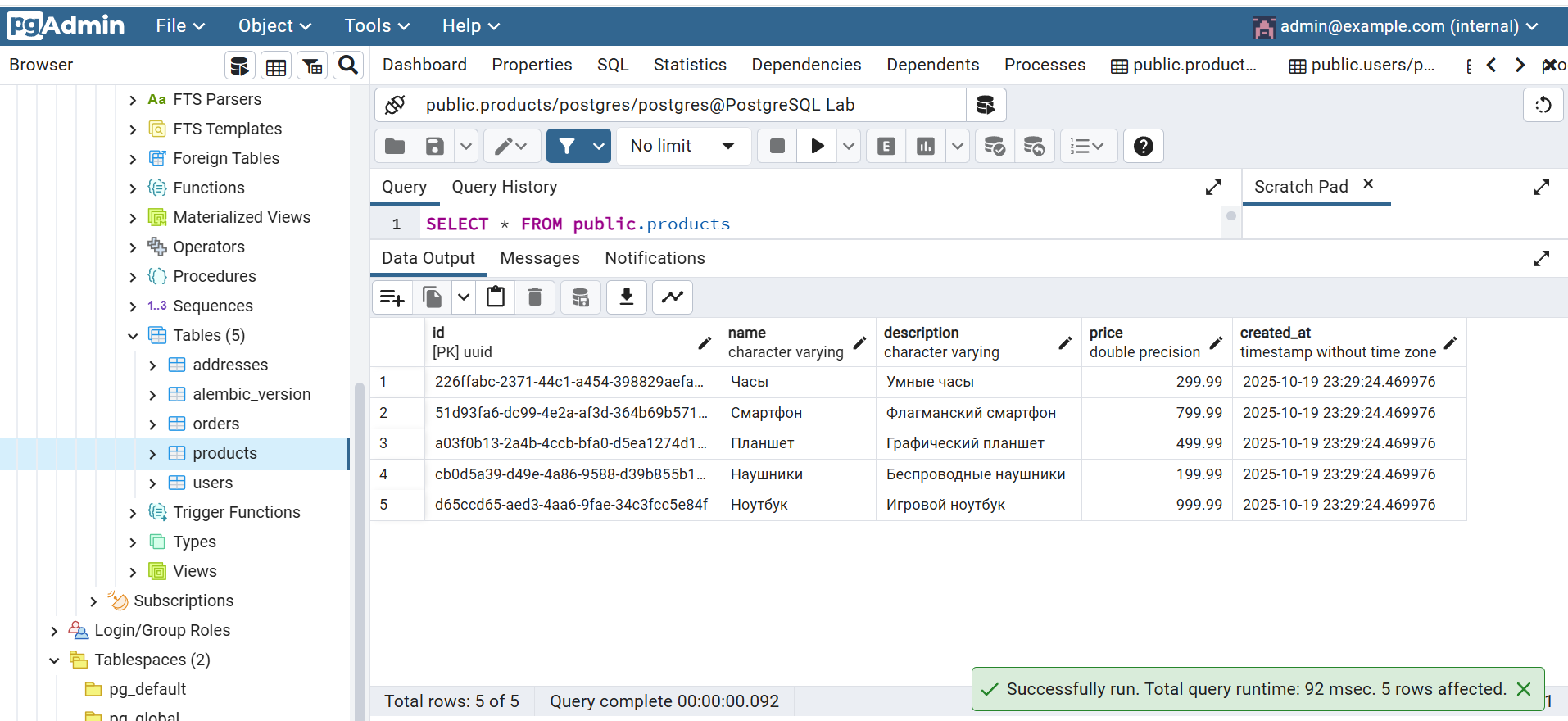
from sqlalchemy import create\_engine  
from sqlalchemy.orm import sessionmaker  
from models import User, Address, Product, Order  
  
engine = create\_engine(  
 "postgresql://postgres:postgres@192.168.99.100:5432/postgres",  
 echo=True  
)  
  
Session = sessionmaker(bind=engine)  
  
print("=== ДОБАВЛЯЕМ ПРОДУКТЫ И ЗАКАЗЫ ===")  
  
with Session() as session:  
 # 1. Добавляем описание существующим пользователям  
 print("1. ОБНОВЛЯЕМ ОПИСАНИЯ ПОЛЬЗОВАТЕЛЕЙ:")  
 users = session.query(User).all()  
 descriptions = [  
 "Постоянный клиент",  
 "Новый пользователь",  
 "VIP клиент",  
 "Корпоративный аккаунт",  
 "Тестовый пользователь"  
 ]  
  
 for i, user in enumerate(users):  
 user.description = descriptions[i]  
 print(f" Обновлен: {user.username}: {user.description}")  
  
 # 2. Создаем 5 продуктов  
 print("\n2. СОЗДАЕМ 5 ПРОДУКТОВ:")  
 products\_data = [  
 {"name": "Ноутбук", "description": "Игровой ноутбук", "price": 999.99},  
 {"name": "Смартфон", "description": "Флагманский смартфон", "price": 799.99},  
 {"name": "Наушники", "description": "Беспроводные наушники", "price": 199.99},  
 {"name": "Планшет", "description": "Графический планшет", "price": 499.99},  
 {"name": "Часы", "description": "Умные часы", "price": 299.99}  
 ]  
  
 products = []  
 for product\_data in products\_data:  
 product = Product(  
 name=product\_data["name"],  
 description=product\_data["description"],  
 price=product\_data["price"]  
 )  
 products.append(product)  
 session.add(product)  
 print(f" Создан: {product.name}: ${product.price}")  
  
 # 3. Создаем 5 заказов  
 print("\n3. СОЗДАЕМ 5 ЗАКАЗОВ:")  
  
 # Получаем существующих пользователей и адреса  
 users = session.query(User).all()  
 addresses = session.query(Address).all()  
  
 orders\_data = [  
 {"user": users[0], "address": addresses[0], "product": products[0], "quantity": 1},  
 {"user": users[1], "address": addresses[1], "product": products[1], "quantity": 2},  
 {"user": users[2], "address": addresses[2], "product": products[2], "quantity": 1},  
 {"user": users[3], "address": addresses[3], "product": products[3], "quantity": 1},  
 {"user": users[4], "address": addresses[4], "product": products[4], "quantity": 3}  
 ]  
  
 for order\_data in orders\_data:  
 order = Order(  
 user\_id=order\_data["user"].id,  
 address\_id=order\_data["address"].id,  
 product\_id=order\_data["product"].id,  
 quantity=order\_data["quantity"]  
 )  
 session.add(order)  
 print(  
 f" Создан заказ: {order\_data['user'].username} -> {order\_data['product'].name} x{order\_data['quantity']}")  
  
 # Сохраняем все изменения  
 session.commit()  
  
 print("\nДАННЫЕ УСПЕШНО ДОБАВЛЕНЫ!")  
 print(f" Продуктов: {len(products)}")  
 print(f" Заказов: {len(orders\_data)}")  
 print(f" Пользователей с описаниями: {len(users)}")

**Результат в консоли: **

результат добавления продуктов:  


данные таблицы orders



Данные таблицы products  


Вопросы

1. Какие есть подходы маппинга в SQLAlchemy? Когда следует использовать каждый подход?

SQLAlchemy предлагает два подхода: декларативный (объединяет описание таблицы и модели) для новых проектов и императивный (разделяет таблицу и класс) для работы с legacy-кодом. Декларативный подход рекомендуется как более простой и современный.

1. Как Alembic отслеживает текущую версию базы данных?

Alembic хранит текущую версию БД в специальной таблице alembic\_version. При выполнении миграций инструмент сверяется с этой таблицей, чтобы определить, какие изменения нужно применить или откатить..

1. Какие типы связей между таблицами вы реализовали в данной работе?

В рамках данной работы были реализованы и изучены основные типы связей между таблицами. На практике была опробована связь «один-ко-многим» (One-to-Many), которая является наиболее распространенной. Также рассмотрены принципы организации связей «один-к-одному» (One-to-One) и «многие-ко-многим» (Many-to-Many), последняя из которых требует создания вспомогательной таблицы для хранения соответствий.

1. Что такое миграция базы данных и почему она важна?

Миграция базы данных — это контролируемое изменение её структуры (например, добавление таблиц или столбцов). Она важна потому, что позволяет безопасно изменять схему БД, отслеживать историю изменений и гарантировать одинаковое состояние базы данных у всех разработчиков и на всех серверах.

1. Как обрабатываются отношения многие-ко-многим в SQLAlchemy?

Отношения «многие-ко-многим» в SQLAlchemy обрабатываются через вспомогательную таблицу, которая хранит связи между двумя основными таблицами. В моделях для этого используется параметр secondary в relationship, который ссылается на эту таблицу-посредник.

1. Каков порядок действий при возникновении конфликта версий в Alembic?

При конфликте версий в Alembic нужно создать миграцию слияния командой alembic merge, проверить сгенерированный код и выполнить alembic upgrade head. Важно не редактировать таблицу alembic\_version вручную, а использовать штатный механизм слияния.

Вывод: d ходе выполнения лабораторной работы были освоены принципы работы с ORM SQLAlchemy и системой миграций Alembic. Были созданы реляционные базы данных ( users, product, orders,, addresses) PostgreSQL, содержащая связанные сущности: пользователи, адреса, продукты и заказы.