Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Кафедра информационных систем и технологий

**Работа с SQLAlchemy и alembic**

Отчет по лабораторной работе 2

по дисциплине «Разработка приложений»

Студент 17.10.2025 А.С. Смирнов

Дата Подпись студ.

Группа РИМ - 150950

Преподаватель 17.10.2025 Д.И. Кузьмин

Дата Подпись препод.

Екатеринбург 2025

## Цели работы

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

## Часть 1. Инициализация базы данных

### 1 Постановка задачи

В ходе части 1 лабораторной работы необходимо сделать ряд действий:

1. Установить нужные библиотеки;

2. Создать ОРМ для пользователя и адреса, также инициализация миграции;

3. Попытка наполнить БД.

### 2 Установка нужных библиотек

Была выполнена команда, что показано на рисунках 1. С помощью команды были установлены библиотеки для работы с базами данных в Python:

1. sqlalchemy - ORM для работы с базами данных;

2. alembic - инструмент для миграций баз данных;

3. psycopg2-binary - PostgreSQL адаптер для Python;

4. asyncpg - асинхронный PostgreSQL драйвер.

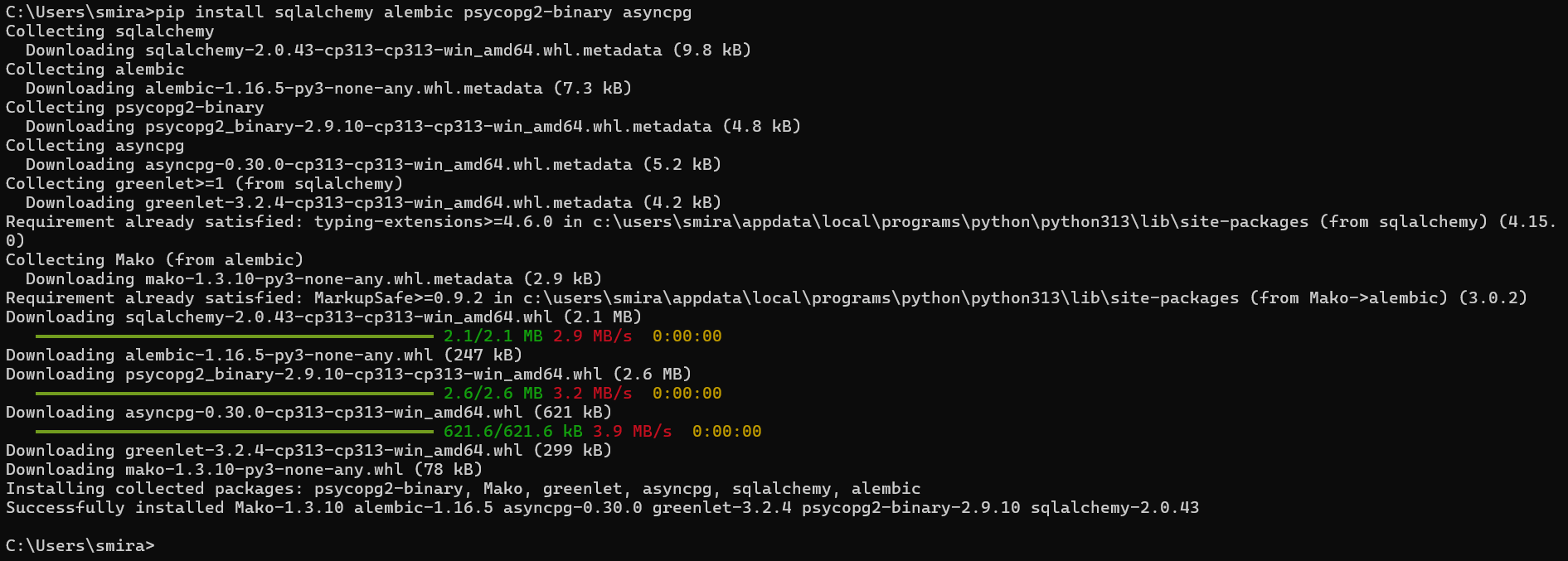


Рисунок 1 – Установка библиотек

### 3 Создание ОРМ для пользователя и адреса, инициализация миграции

Созданная ОРМ для пользователя и адреса показаны на рисунке 2. Инициализация миграции показана на рисунке 3. Результат изменения строки подключения в alembic.ini к БД показан на рисунке 4. В env.py подставляем метаданные для миграции, показано на рисунке 5. Создание миграции и применение к БД последней миграции показано на рисунке 6. Создание фабрики подключения показано на рисунке 7. Наполнение БД данными о пяти пользователях и их адресов показано на рисунке 8.

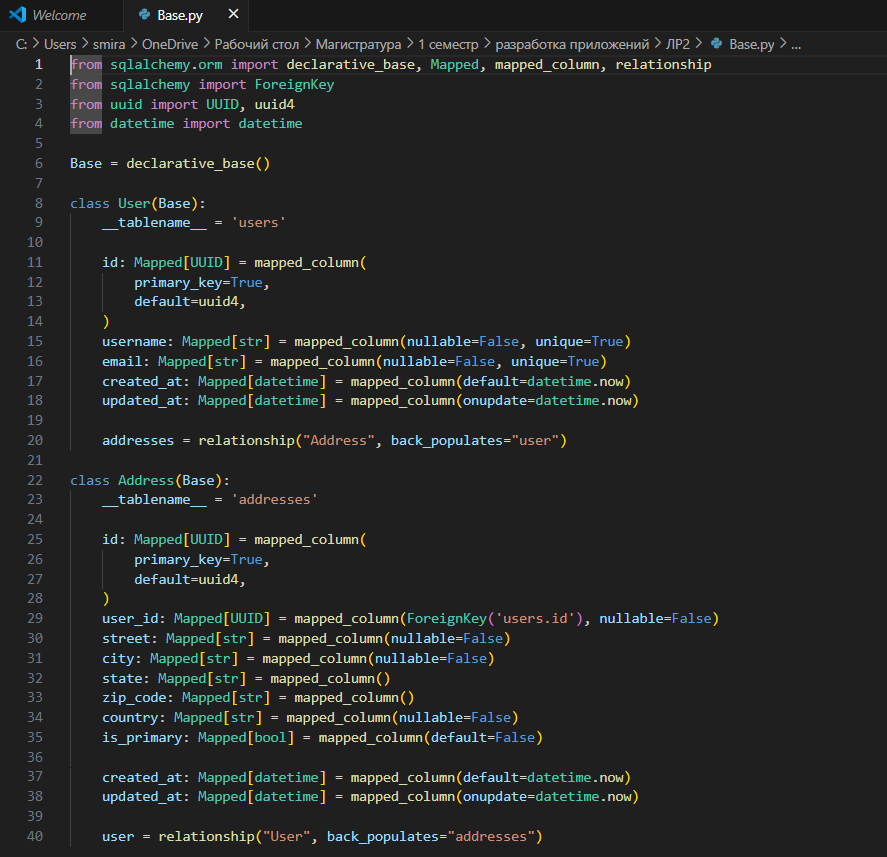


Рисунок 2 - Созданная ОРМ для пользователя и адреса

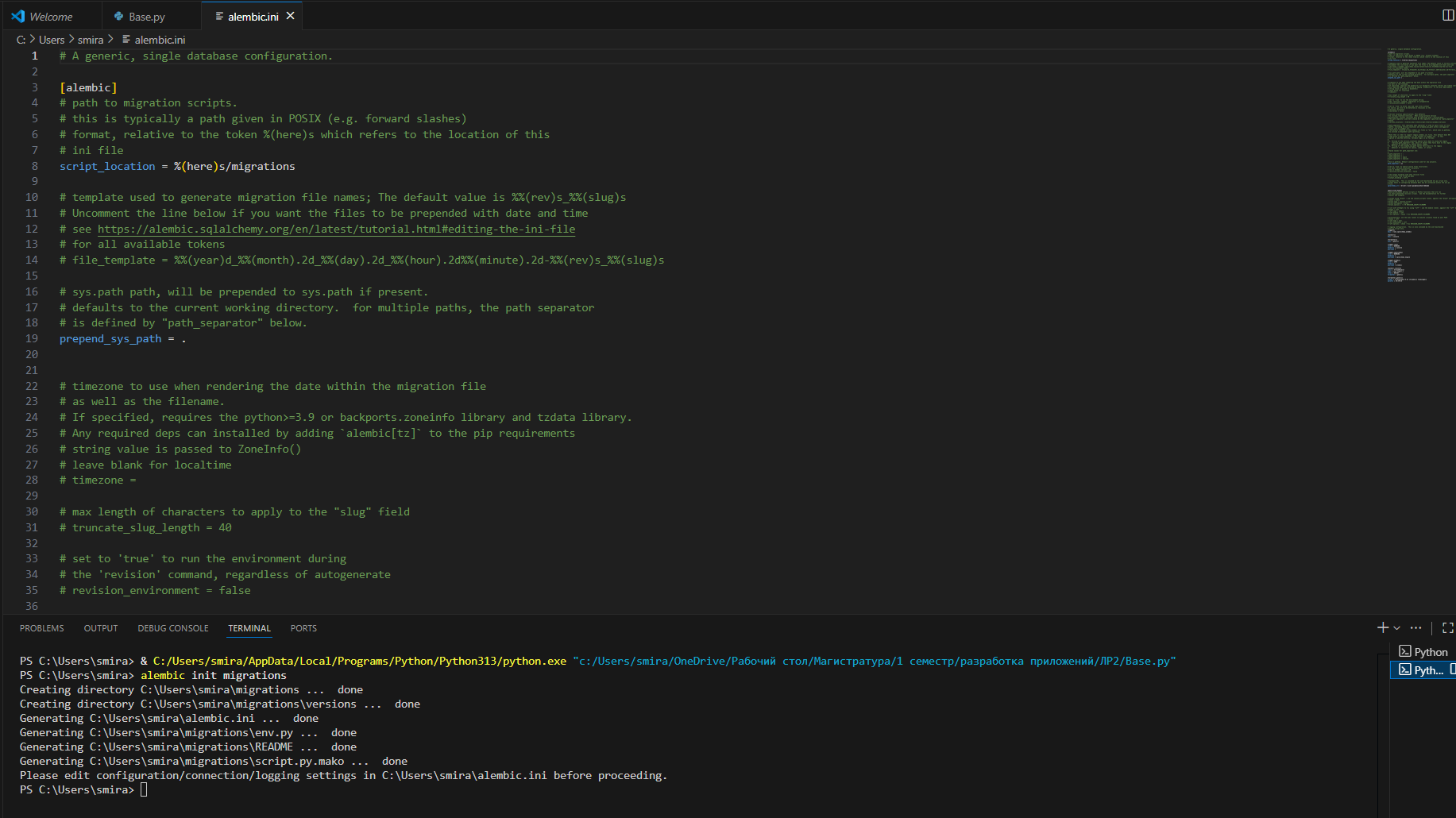


Рисунок 3 – Инициализация миграции

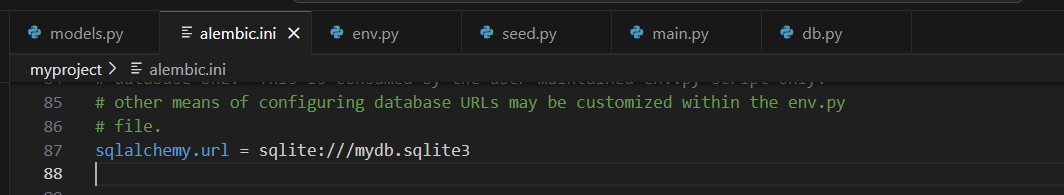


Рисунок 4 – Работа с alembic.ini

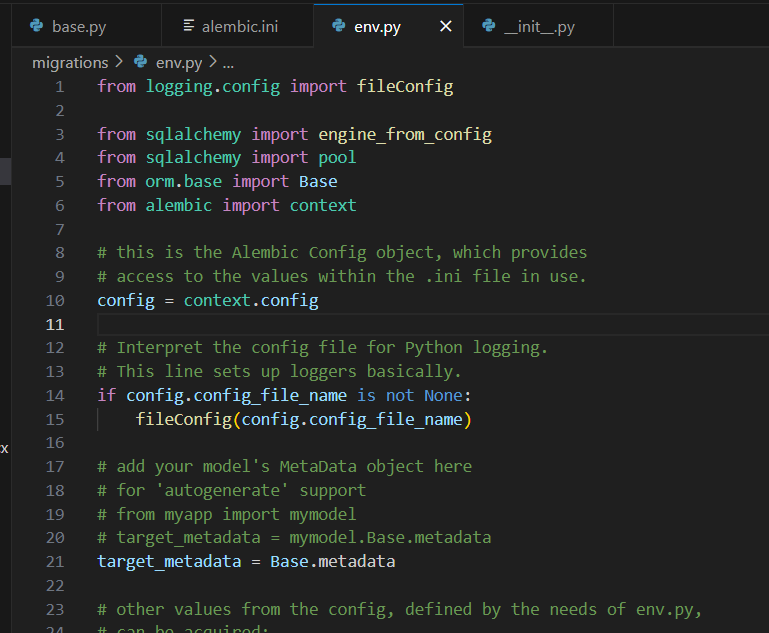


Рисунок 5 – Работа с env.py

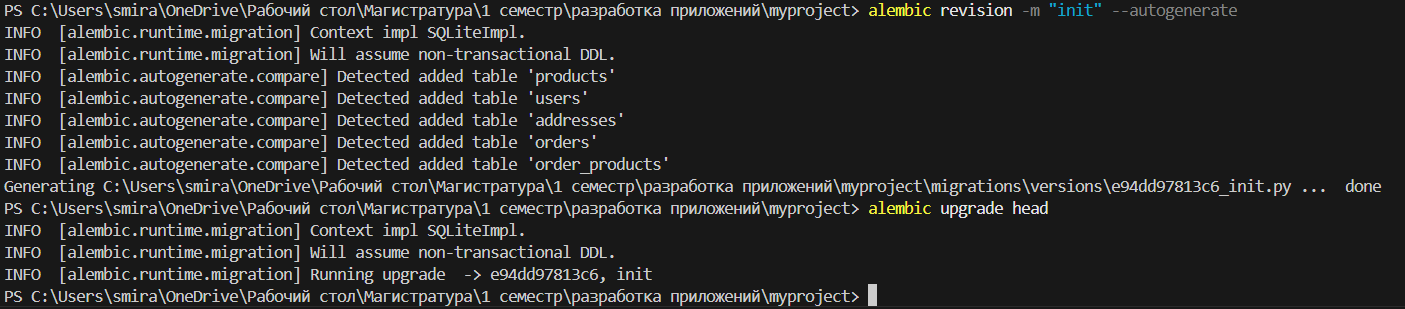


Рисунок 6 – Создание миграции и применение к БД последней миграции

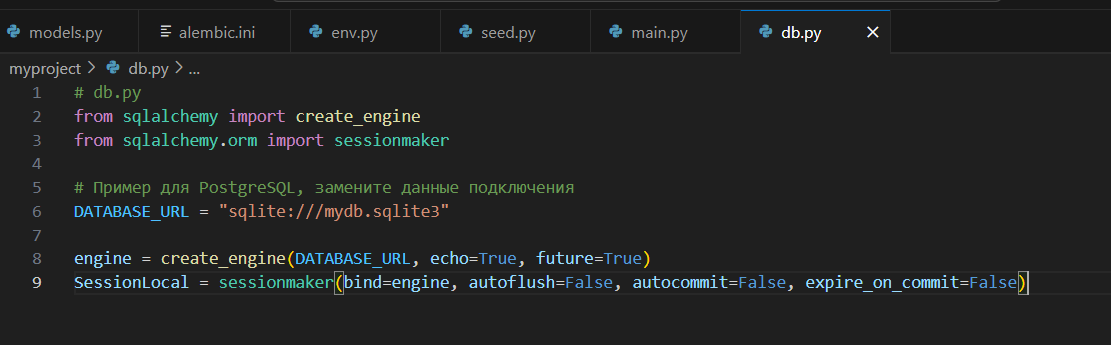


Рисунок 7 – Создание фабрики подключения

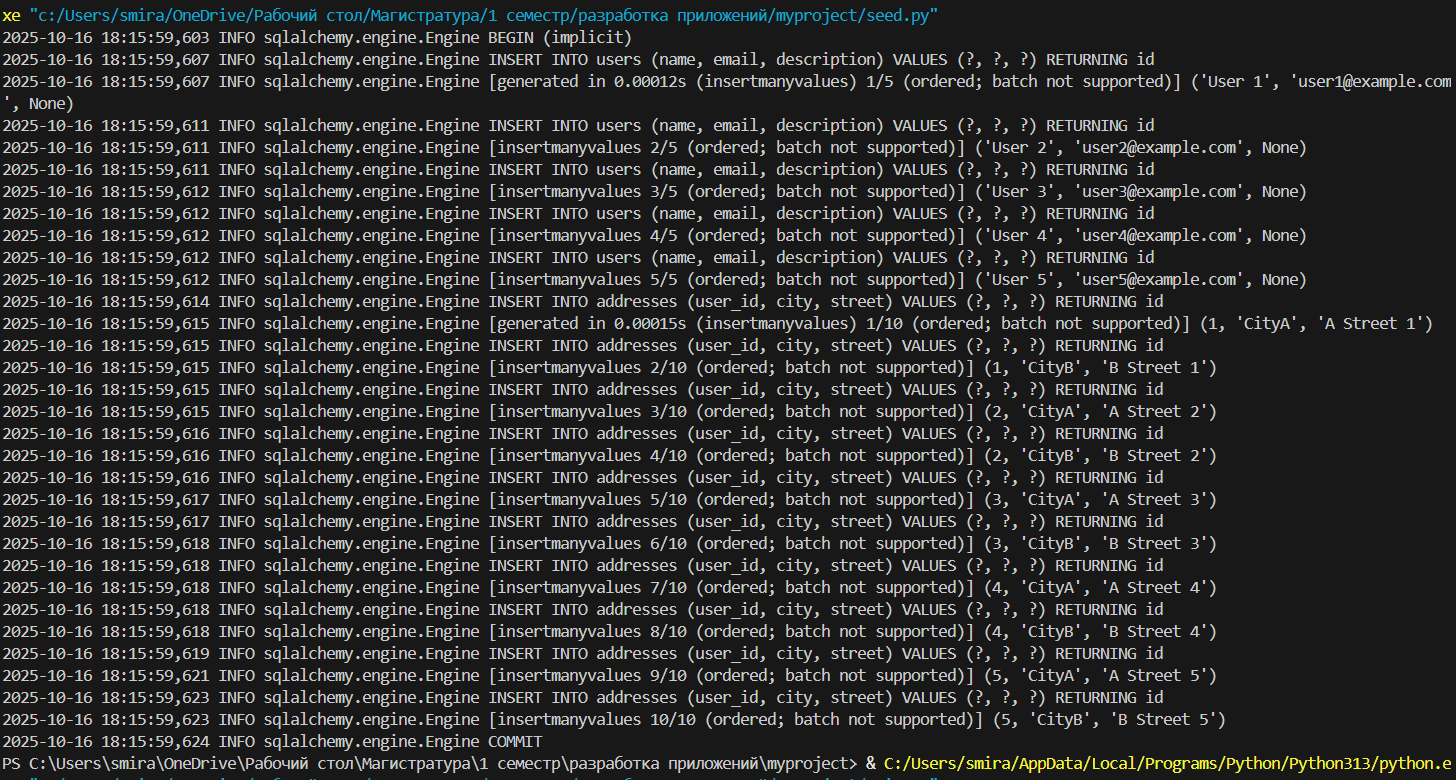


Рисунок 8 – Наполнение БД данными о пяти пользователях и их адресов

## Часть 2: Запрос связанных данных и последующие работы с БД миграции

### 1 Постановка задачи

В ходе части 2 лабораторной работы необходимо выполнить:

1. Получить и вывести данные из двух таблиц;

2. Указать дополнительные параметры запроса;

3. Указать дополнительные данные, которые нужно получить в запросе;

4. Добавить пользователю дополнительное строковое поле description;

5. Добавить дополнительную таблицу для продукции и заказов;

6. Заказ должен в себе содержать информацию о пользователе, адресе доставки и продукции;

7. Производиться миграция данных и добавляется 5 продукций и 5 заказов в БД.

### 2 Выполнение запроса связанных данных

Результат выполнения запроса связанных данных, с изменениями в коде, показан на рисунке 9.

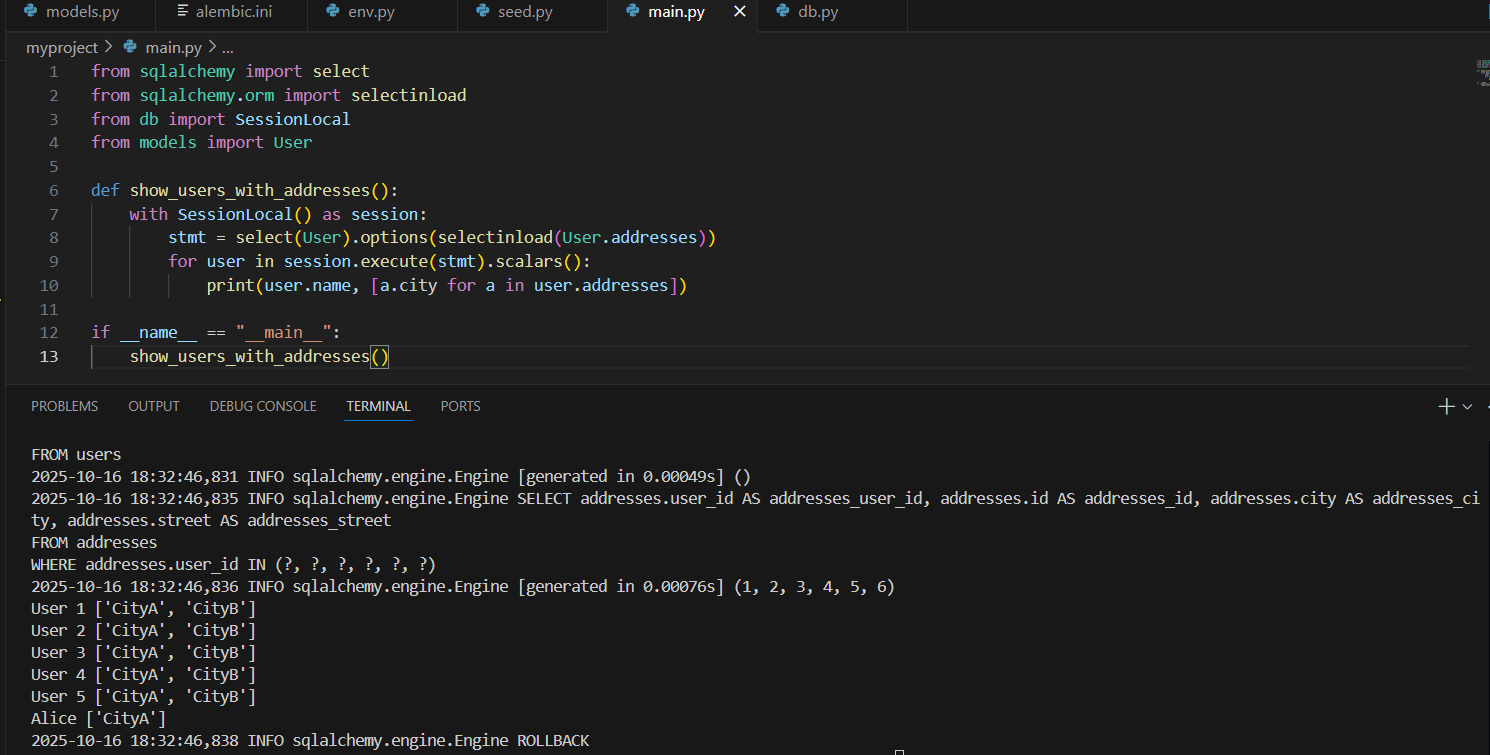


Рисунок 9 – Результат выполнения запроса связанных данных

### 3 Последующие работы с БД и миграции

В ОРМ добавляется пользователю дополнительное строковое поле description, что показано на рисунке 10. Результат проведенной миграции показан на рисунке 11. Добавление функций для создания пяти товаров и пяти заказов показано на рисунке 12. С помощью команды python -c "from seed import seed\_products\_and\_orders; seed\_products\_and\_orders()" были созданы товары и заказы, что показано на рисунке 13.

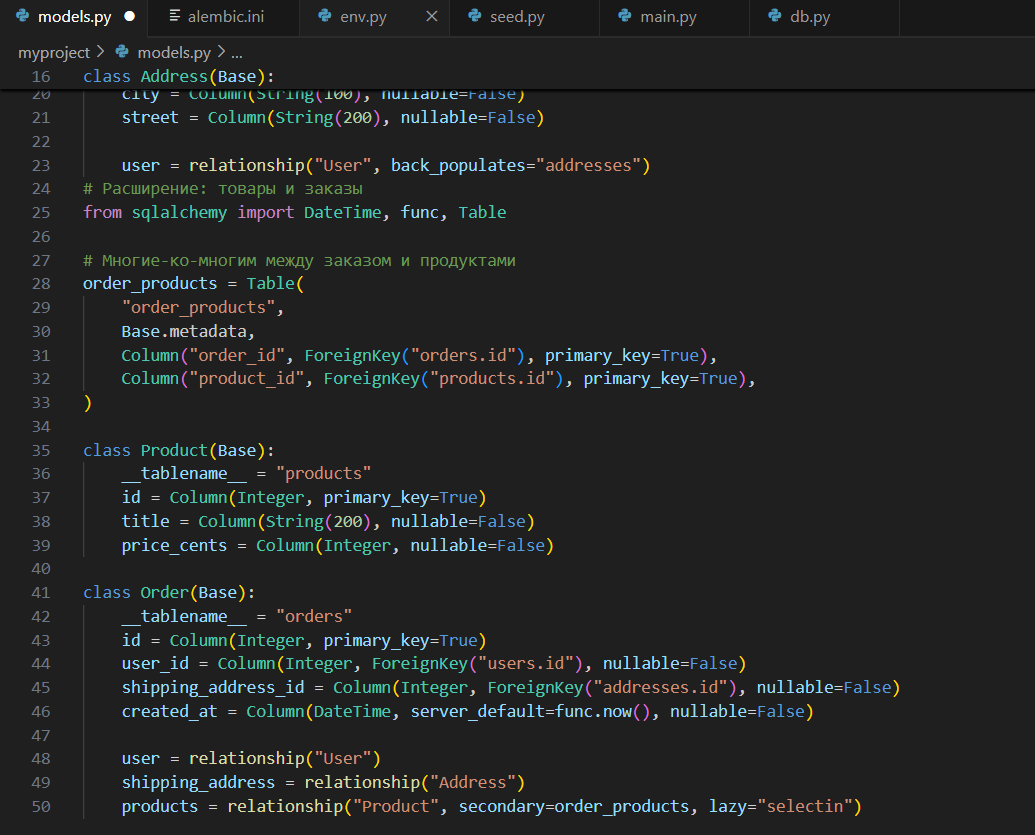


Рисунок 10 – Добавление пользователю строкового поля description

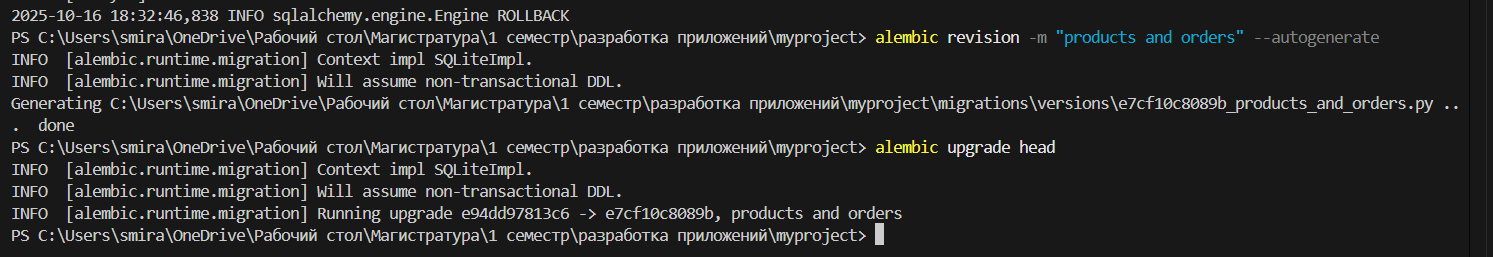


Рисунок 11 – Результат проведенной миграции

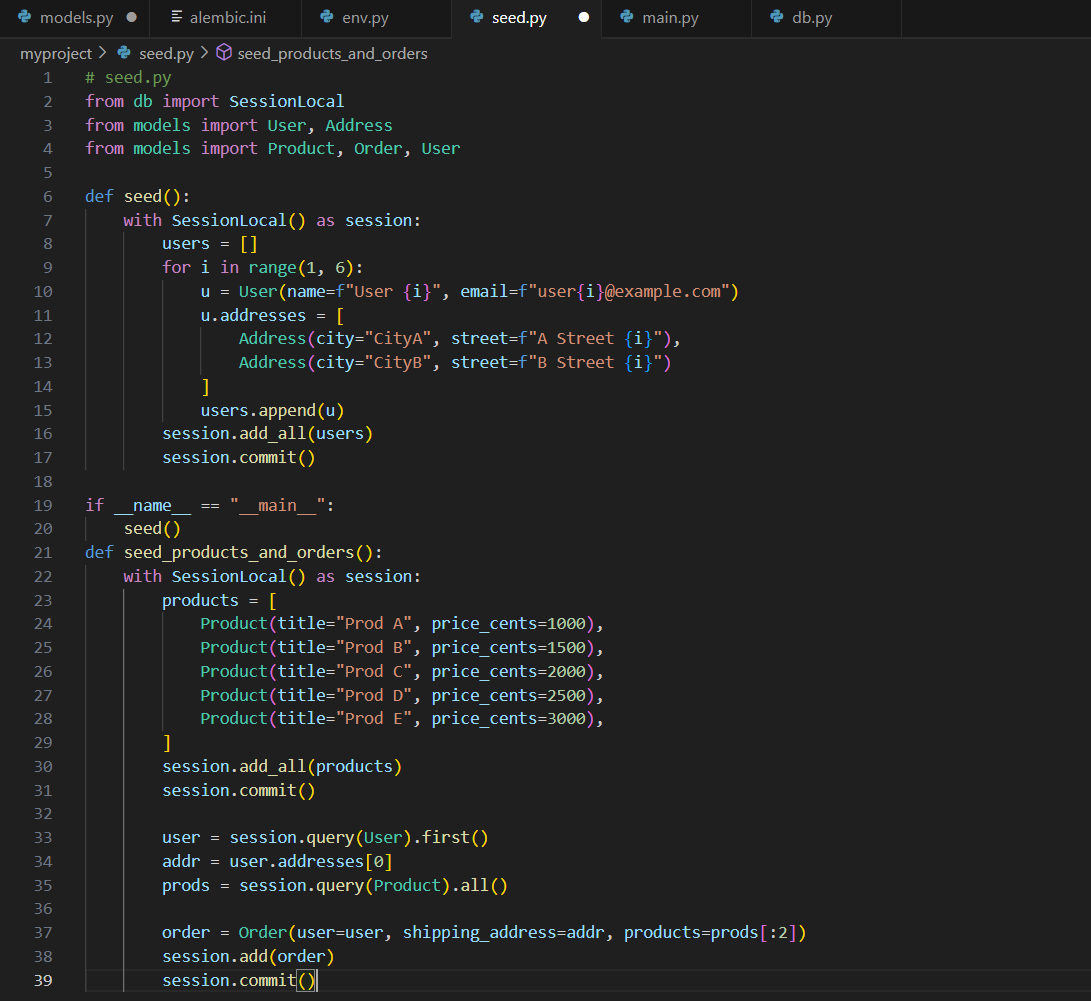


Рисунок 12 – Добавление функций для создания пяти товаров и пяти заказов

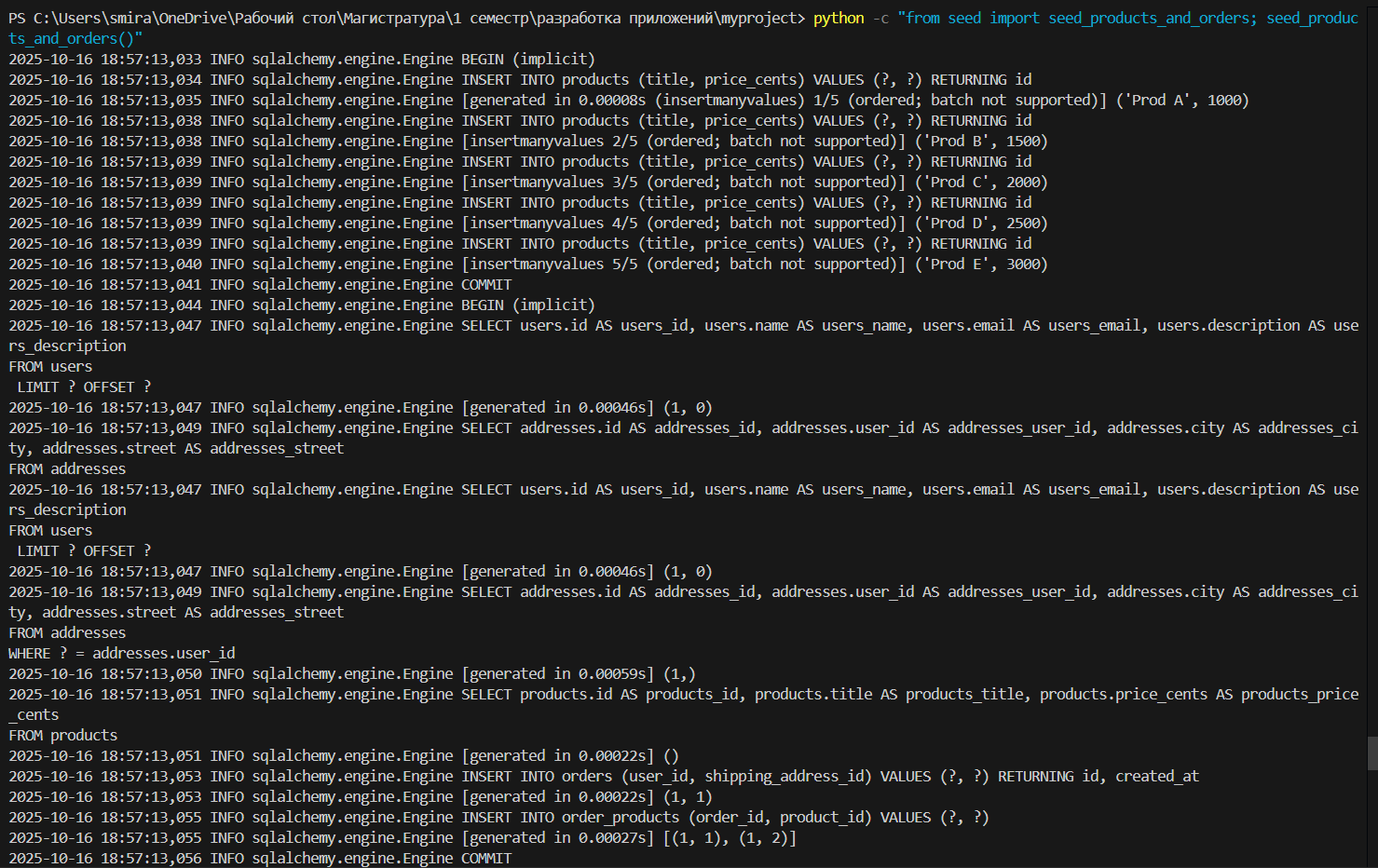


Рисунок 13 – Добавление функций для создания пяти товаров и пяти заказов

## ВЫВОД

В ходе работы была проведена работа инициализации базы данных с установкой нужных библиотек, была создана ОРМ для пользователей и адресов, также была проведена инициализация миграции.

Был проведен запрос связанных данных и последующие работы с БД, а также миграции.

**1. Подходы маппинга в SQLAlchemy**

В SQLAlchemy есть три основных способа маппинга классов Python на таблицы базы данных:

а) Декларативный маппинг - самый популярный способ, использует наследование от базового класса, определённого через declarative\_base(). Обычно рекомендуется для современных приложений из-за простоты и удобочитаемости.

б) Классический маппинг - вручную связывает объекты Table и Python-классы через функцию mapper(). Подходит для сложных или нестандартных ситуаций, когда нужен полный контроль над процессом.​

в) Ассоциативный маппинг - сходен с классическим, но использует методы класса и подклассифицирование, подходит для гибких схем или совместимости с существующим кодом.

Используется в следующих случаях:

а) Декларативный маппинг подходит для большинства случаев, когда удобно описывать схему базы в виде классов Python.

б) Классический маппинг стоит выбирать для работы с сильно динамическими или существующими структурами.

в) Ассоциативный маппинг - для продвинутых сценариев интеграции или специфических нужд.

**2. Как Alembic отслеживает текущую версию базы данных?**

Alembic создаёт специальную таблицу alembic\_version в базе данных. Эта таблица содержит единственную строку - идентификатор последней успешно применённой миграции (version\_num). При запуске Alembic сверяет содержимое этой таблицы с вашей историей миграций, чтобы определить, какие изменения были выполнены и какие ещё нужно применить.

**3. Какие типы связей между таблицами реализуют?**

В реляционных базах данных (и в SQLAlchemy) есть три основных типа связей:

1) Один-к-одному (One-to-One)

2) Один-ко-многим (One-to-Many)

3) Многие-ко-многим (Many-to-Many)

**4. Что такое миграция базы данных и почему она важна?**

Миграция базы данных - это контроль и применение последовательных изменений структуры базы (например, добавление столбцов, таблиц, ключей). Миграции позволяют синхронизировать структуру базы с изменениями кода, поддерживают версионность и историю изменений, предохраняют от потери или нарушения данных при обновлениях.​

**5. Как обрабатываются отношения многие-ко-многим в SQLAlchemy?**

Связь "многие-ко-многим" реализуется через вспомогательную/связующую таблицу - она содержит по внешнему ключу на каждую из связанных таблиц. В SQLAlchemy это обычно делается с помощью необъявленной в ORM вспомогательной таблицы (Table) и указания её в параметре secondary при создании отношений с relationship.

**6. Каков порядок действий при возникновении конфликта версий в Alembic?**

Конфликт версий возникает, если миграции расходятся (например, несколько веток миграций "разошлись" и обе были применены). Обычно порядок действий такой:

1) Остановить все работы с БД.

2) Выяснить и визуализировать, где произошла "развилка" (например, используя alembic history --verbose).

3) Написать объединяющую (merge) миграцию через команду alembic merge, чтобы свести разошедшиеся ветви в одну.

4) Проверить корректность результата, протестировать изменения.

5) Зафиксировать и синхронизировать состояние всех сред, чтобы повторно не возник конфликт.