Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий-РТФ

ОТЧЕТ

По лабораторной работе №2

«Работа с SQLAlchemy и alembic»

По дисциплине «Разработка приложений»

Выполнил: Гуламов А.С.

Группа: РИМ-150950

Проверил преподаватель: Кузьмин Д.

Екатеринбург

2025

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

**Задачи:**

1. Настроить рабочее окружение.
2. Разработать ORM-модели данных с использованием SQLAlchemy.
3. Настроить и использовать систему миграций Alembic.
4. Реализовать работу с данными через ORM
5. Проверить корректность работы системы

**Ход работы:**

**1. Подготовка среды разработки**

В VS Code создаем папку проекта, создаем и активируем виртуальное окружение, загружаем требуемые библиотеки с помощью команд (Рисунок 1):

* mkdir LR2;
* cd LR2;
* python -m venv venv;
* .\venv\Scripts\activate.
* pip install sqlalchemy alembic psycopg2-binary

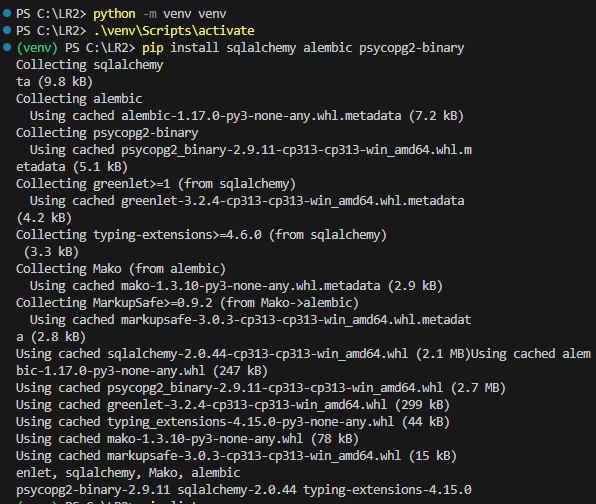


Рисунок 1 – Подготовка

Далее запускаем PostgreSQL в Docker и убеждаемся в работе (Рисунок 2).

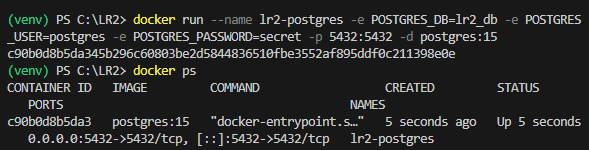


Рисунок 2 – PostgreSQL в Docker

**2. Создание и настройка файлов**

Создали файл моделей models.py (файл прикреплен в репозитории). Далее инициализировали Alembic с помощью команды «python -m alembic init migrations», появились папка «migrations» и файл «alembic.ini» (Рисунок 3)

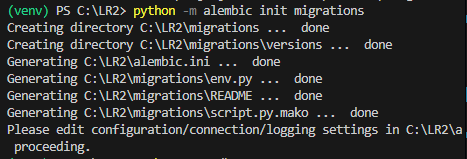


Рисунок 3 – Запуск команды «python -m alembic init migrations»

В файле «alembic.ini» нашли строку:

* «sqlalchemy.url = driver://user:pass@localhost/dbname»

и заменили ее на

* «sqlalchemy.url =postgresql://postgres:secret@localhost:5432/lr2\_db»

В файле «migrations/env.py» нашли строку:

* «target\_metadata = None»

и заменили ее на:

* «from models import Base»
* «target\_metadata = Base.metadata»

Далее создали и применили миграцию с помощью команд (Рисунок 4):

* «python -m alembic revision --autogenerate -m "init tables"»
* «python -m alembic upgrade head»

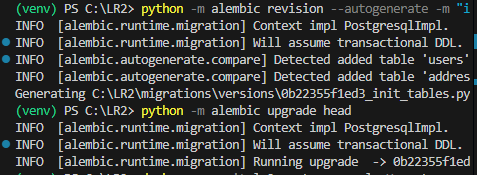


Рисунок 4 – Работа с миграциями

Проверили корректность работы (Рисунок 5).

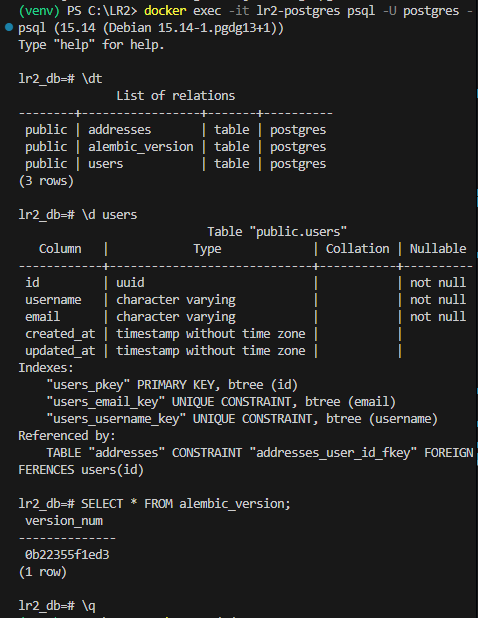


Рисунок 5 – Подключение к БД внутри контейнера

**3. Наполнение БД данными**

Создадим скрипт для наполнения БД «seed\_data.py» (файл прикреплен в репозитории) и запустим его командой «python seed\_data.py», в результате чего БД наполнилась 5 пользователями и их адресами. Проверим корректность работы (Рисунок 6).

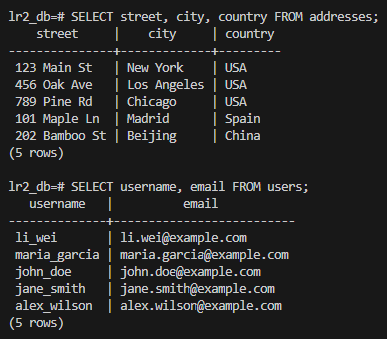


Рисунок 6 – Подключение к БД внутри контейнера для просмотра наполнения БД

**4. Запрос связанных данных**

Создали файл «query\_data.py» (файл прикреплен в репозитории) и запустим его командой «python query\_data.py» (Рисунок 7).

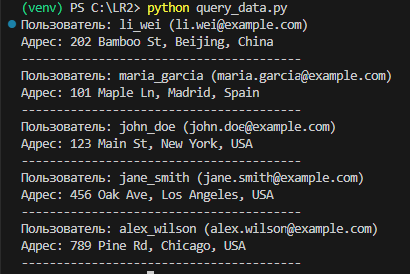


Рисунок 7 – Вывод данных о каждом пользователе

**5. Модификация БД**

В файле «models.py» добавили поле «description» в класс User (Рисунок 8). А также добавили новые таблицы «Product» и «Order» и чуть не забыли про связи! (Рисунок 9).

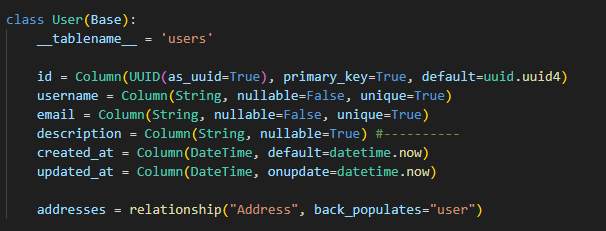


Рисунок 8 – Добавление поля «description»



Рисунок 9 – Добавление таблиц «Product» и «Order»

Далее, чтобы сгенерировать миграцию, прописали команду «python -m alembic revision --autogenerate -m "add description to user, add products and orders"» и применили миграцию командой «python -m alembic upgrade head»

Наполним БД новыми данными, создав файл «seed\_more\_data.py» и запустив его командой «python seed\_more\_data.py». Проверим изменения (Рисунок 10).

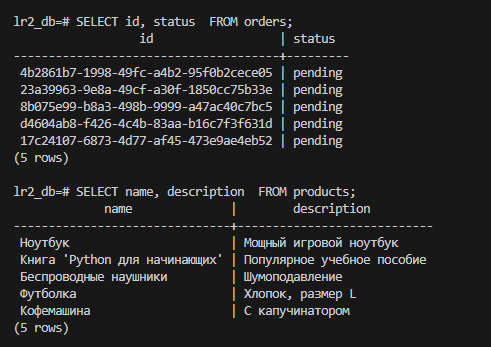


Рисунок 10 – Новые данные

**Вопросы:**

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

* Декларативный - самый простой: требует написания класса, после этого всё сразу понятно.
* Классический - сначала нужно описать таблицу, потом класс, потом связать их.
* Автомаппинг - когда имеется база, а SQLAlchemy сам придумывает классы

В данной работе мы использовали первый, потому что он самый лёгкий.

2. Как Alembic отслеживает текущую версию базы данных?

Alembic создаёт в базе специальную служебную таблицу под названием «alembic\_version». В ней хранится одна строчка - аналог «ярлыка», который хранит ID последней применённой миграции. При запуске «alembic upgrade head», Alembic смотрит, какая версия сейчас в таблице, какие миграции есть в папке versions и применяет только те, которых ещё нет.

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

Мы сделали связь «один-ко-многим»: один пользователь может иметь несколько адресов, но каждый адрес принадлежит только одному пользователю. В таблице «addresses» есть поле «user\_id», которое ссылается на «id» в таблице «users». В коде мы использовали ForeignKey и relationship, чтобы SQLAlchemy знал, как эти таблицы связаны. Связи «многие ко многим» (например, один пользователь - много продуктов, и один продукт - у многих пользователей) у нас не было, но можно было бы добавить, если бы за один заказ можно было бы купить несколько товаров.

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

Миграция - файл с инструкциями, как изменить структуру базы данных: добавить таблицу, колонку, изменить тип и т.д. Она важна, потому что без неё пришлось бы вручную редактировать базу на каждом компьютере, а с миграциями же все изменения сохраняются в коде. В случае ошибки можно откатиться назад.

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

Связь «многие ко многим» (например, «пользователь может покупать много товаров, и товар может быть у многих пользователей») нельзя сделать напрямую в реляционной базе. Нужна дополнительная таблица-посредник.

**Вывод:**

В ходе выполнения лабораторной работы были освоены основы работы с библиотеками SQLAlchemy и Alembic для управления реляционной базой данных в Python.

Были созданы ORM-модели пользователей, адресов, продуктов и заказов, настроены связи между ними и запущены PostgreSQL в Docker без установки СУБД на компьютер.

С помощью Alembic я были сгенерированы и применены миграции, что позволило безопасно изменять структуру базы данных без ручного вмешательства.

При наполнении БД тестовыми данными и выполнении запросов к связанным таблицам убедились, что всё работает корректно.

Работа показала, насколько удобно и надёжно можно управлять базой данных через код: легко воспроизводятся и не зависят от конкретного окружения. Полученные навыки могут быть использованы в будущих проектах для построения гибких и поддерживаемых приложений.

**Ссылка на репозиторий:**

**https://github.com/bepis-art/application-development-2025-urfu.git**