

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Лабораторная работа №2
Работа с SQLAlchemy и alembic

Студент группы РИМ – 150950: _____ Вальнева А.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. Подготовить окружение для работы с базой данных, установив необходимые библиотеки.
2. Проектирование и реализация ORM-моделей.
3. Работа с системой миграций Alembic.
4. Применение миграций и управление версиями БД
5. Создание CRUD-операций и тестовых данных

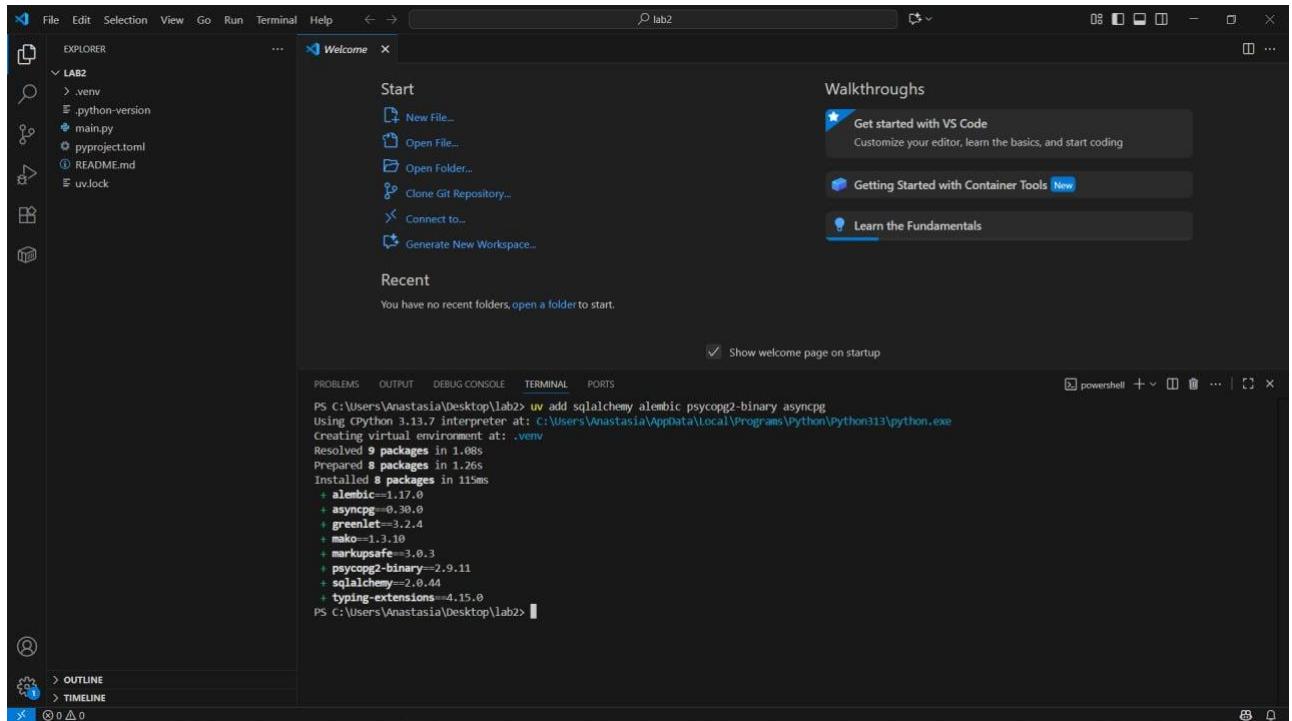
Структура проекта

Lab2/

```
├── alembic.ini          # Конфигурация Alembic
├── migrations/
│   ├── versions/
│   └── env.py |
|
└── db/                  # Папка с моделями и базовым классом
    ├── models/
    │   ├── user.py        # Модель пользователя
    │   ├── address.py     # Модель адреса
    │   ├── order.py       # Модель заказа
    │   └── product.py     # Модель продукта
    ├── base.py           # Базовый класс моделей
    └── session.py        # Настройки сессии БД
|
└── crud/
    ├── user.py          # Ф-ии для работы с пользователями
    ├── address.py        # Файл для работы с адресами
    ├── order.py
    └── product.py
|
└── main.py            # Главный скрипт
```

Часть 1: Инициализация БД

Задача: подготовить окружение для работы с базой данных, установив необходимые библиотеки



Создаем новый проект с помощью команды **uv init**
В терминале запускаем команду установки пакетов

- **uv add sqlalchemy alembic psycopg2-binary asynccpg**

Установлено 8 пакетов в виртуальное окружение (файл uv.lock обновлен; создана папка .venv для изолированного окружения).

```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 lab2
EXPLORER
LAB2
> .venv
python-version
main.py
models.py
pyproject.toml
README.md
uv.lock
OUTLINE
TIMELINE
models.py
1 from sqlalchemy import (
2     String,
3     Boolean,
4     ForeignKey,
5     DateTime,
6 )
7 from sqlalchemy.orm import (
8     declarative_base,
9     Mapped,
10    mapped_column,
11    relationship,
12 )
13 from uuid import uuid4
14 from datetime import datetime
15
16 Base = declarative_base()
17
18 class User(Base):
19     __tablename__ = 'users'
20
21     id: Mapped[str] = mapped_column(
22         primary_key=True,
23         default=lambda: str(uuid4()))
24
25     username: Mapped[str] = mapped_column(nullable=False, unique=True)
26     email: Mapped[str] = mapped_column(nullable=False, unique=True)
27     created_at: Mapped[datetime] = mapped_column(default=datetime.now)
28     updated_at: Mapped[datetime] = mapped_column(onupdate=datetime.now)
29
30     addresses = relationship("Address", back_populates="user")
31
32
33 class Address(Base):
34     __tablename__ = 'addresses'
35
36     id: Mapped[str] = mapped_column(
37         primary_key=True,
38         default=lambda: str(uuid4()))
39
Ln 53, Col 1  Spaces: 4  UTF-8  CRLF  Python  ⌂
```

Задача: определить структуру таблиц БД через ORM-модели для пользователя и адреса.

Создали файл models.py

Импорты:

```
from sqlalchemy import String, DateTime, ForeignKey
from sqlalchemy.orm import declarative_base, Mapped, mapped_column, relationship
from uuid import uuid4
from datetime import datetime
```

Импорты из SQLAlchemy ORM

- declarative_base (фабрика для создания базового класса моделей)
- Mapped и mapped_column (для аннотаций полей)
- Relationship (определение связей между моделями (отношения))

Стандартные импорты Python

- uuid4 (для генерации идентификаторов)
- datetime из datetime (работа с датой и временем)

Объявили 2 класса: User и Address (определяем базовый класс с помощью declarative_base())

```

models.py
1  from sqlalchemy import (
2      String,
3      Boolean,
4      ForeignKey,
5      DateTime,
6  )
7  from sqlalchemy.orm import (
8      declarative_base,
9      Mapped,
10     mapped_column,
11     relationship,
12  )
13 from uuid import uuid4
14 from datetime import datetime
15
16 Base = declarative_base()
17
18
● (lab2) PS C:\Users\Anastasia\Desktop\lab2> alembic init migrations
Creating directory C:/Users/Anastasia/Desktop/lab2/migrations ... done
Creating directory C:/Users/Anastasia/Desktop/lab2/migrations/versions ... done
Generating C:/Users/Anastasia/Desktop/lab2/alembic.ini ... done
Generating C:/Users/Anastasia/Desktop/lab2/migrations/env.py ... done
Generating C:/Users/Anastasia/Desktop/lab2/migrations/README ... done
Generating C:/Users/Anastasia/Desktop/lab2/migrations/script.py.mako ... done
Please edit configuration/connection/logging settings in C:/Users/Anastasia/Desktop/lab2/alembic.ini before proceeding.
○ (lab2) PS C:\Users\Anastasia\Desktop\lab2>

```

Инициализация миграций Alembic

- **alembic init migrations.**

Создана папка migrations с необходимыми файлами.

```

[sqlalchemy.url] = postgresql+psycopg2://user:superpass@localhost:5432/my_db

```

С помощью docker compose развернута БД Postgres из лабораторной работы 1.

В файле alembic.ini в корне проекта меняем строку подключения к БД:

- `sqlalchemy.url = postgresql+psycopg2://user:superpass@localhost:5432/my_db`

```

1  from logging.config import fileConfig
2
3  from sqlalchemy import engine_from_config
4  from sqlalchemy import pool
5
6  from alembic import context
7
8  from db.base import Base
9  from db import models
10
11 # this is the Alembic Config object, which provides
12 # access to the values within the .ini file in use.
13 config = context.config
14
15 # Interpret the config file for Python logging.
16 # this line sets up loggers basically.
17 if config.config_file_name is not None:
18     fileConfig(config.config_file_name)
19
20 # add your model's MetaData object here
21 # for 'autogenerate' support
22 # from myapp import mymodel
23 # target_metadata = mymodel.Base.metadata
24 target_metadata = Base.metadata
25
26 # other values from the config, defined by the needs of env.py,
27 # can be acquired:
28 # my_important_option = config.get_main_option("my_important_option")
29 # ... etc.
30
31
32 def run_migrations_offline() -> None:
33     """Run migrations in 'offline' mode.
34
35     This configures the context with just a URL
36     and not an Engine, though an Engine is acceptable
37     here as well. By skipping the Engine creation
38     we don't even need a DBAPI to be available.
39
40     Calls to context.execute() here emit the given string to the

```

Line 21, Col 29 Spaces: 4 UTF-8 LF ⚙ Python

Импорты:

```

from db.base import Base          # Для метаданных
from db import models             # Для обнаружения всех моделей

```

Установлены метаданные для миграций:

- target_metadata = Base.metadata

```

PS C:\Users\Anastasia\Desktop\lab2> alembic revision --autogenerate -m "create user and address tables"
INFO [alembic.runtime.migration] context impl PostgresImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'users'
INFO [alembic.autogenerate.compare] Detected added table 'addresses'
Generating C:\Users\Anastasia\Desktop\lab2\migrations\versions\5fb292750b0b_create_user_and_address_tables.py ...
done

```

Создана первая миграция

- alembic revision --autogenerate -m "create user and address tables"

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following output:

```
(lab2) PS C:\Users\nastasia\Desktop\lab2> alembic upgrade head
INFO [alembic.runtime.migration] Context impl PostgresImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade -> 5fb292750b0b, create user and address tables
```

Миграция применена к БД:

- alembic upgrade head

Задача: наполнить БД данными

1) Настройка подключения к базе данных (db/ session.py)

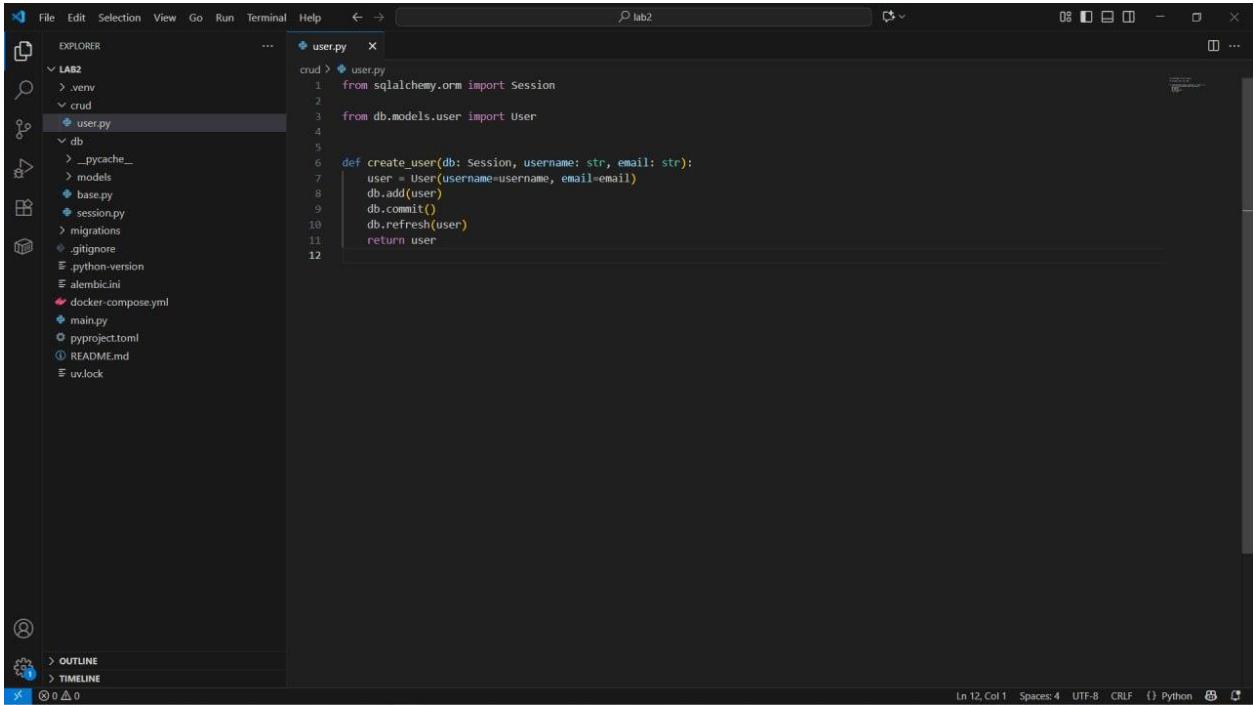
The screenshot shows the VS Code interface with the code editor tab active. The editor window displays the `session.py` file content:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from db.base import Base
DATABASE_URL = "postgresql+psycopg2://user:superpass@localhost:5432/my_db"
engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

- В `session.py` задается строка подключения к базе данных (`DATABASE_URL`)

- Создается фабрика подключений
 - engine = create_engine(DATABASE_URL, echo=True)
- SessionLocal - фабрика сессий для работы с БД

2) Функция создания пользователя (crud/user.py)



```

File Edit Selection View Go Run Terminal Help ⏎ → 🔍 lab2
EXPLORER crud user.py
LAB2
  > .venv
  > crud
    & user.py
  > db
    > __pycache__
    > models
      base.py
      session.py
  > migrations
  & .gitignore
  & python-version
  & alembic.ini
  docker-compose.yml
  main.py
  pyproject.toml
  README.md
  uv.lock

user.py
crud > user.py
1  from sqlalchemy.orm import Session
2
3  from db.models.user import User
4
5
6  def create_user(db: Session, username: str, email: str):
7      user = User(username=username, email=email)
8      db.add(user)
9      db.commit()
10     db.refresh(user)
11
12     return user

```

Ln 12, Col 1 Spaces: 4 UTF-8 CRLF {} Python

Функция create_user:

- принимает сессию базы данных, имя пользователя и email;
- создает объект пользователя, добавляет его в сессию (db.add(user)), сохраняет в БД (db.commit()) и обновляет объект из БД (db.refresh(user)) ;
- возвращает созданного пользователя;

3) Файл main.py

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under the "LAB2" folder, including files like user.py, db, migrations, and docker-compose.yml.
- Editor:** Displays the main.py file with the following code:

```
from db.session import SessionLocal
from crud.user import create_user

def main():
    db = SessionLocal()
    try:
        user = create_user(db, username="John Doe", email="jdoe@example.com")
        print(f"Created: {user.username}")
    finally:
        db.close()

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

The status bar at the bottom indicates "Ln 14, Col 11" and "Python".

Функция main():

- Создает сессию БД;
- В блоке try/finally гарантирует закрытие сессии;
- Создает одного тестового пользователя;
- Выводит результат;

4) Функция создания адреса (crud/address.py)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under the "crud" folder, including files like address.py, user.py, db, migrations, and docker-compose.yml.
- Editor:** Displays the address.py file with the following code:

```
from sqlalchemy.orm import Session
from db.models.address import Address

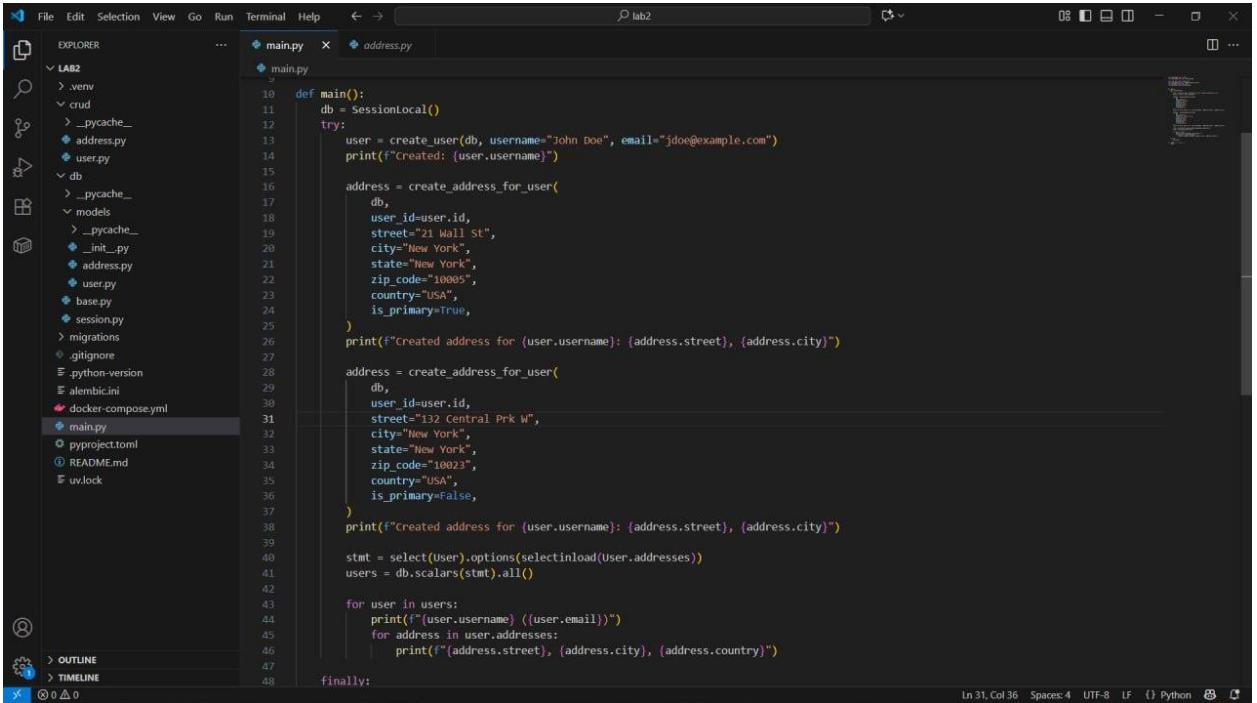
def create_address_for_user(
    db: Session,
    user_id: str,
    street: str,
    city: str,
    country: str,
    state: str = None,
    zip_code: str = None,
    is_primary: bool = False,
):
    address = Address(
        user_id=user_id,
        street=street,
        city=city,
        state=state,
        zip_code=zip_code,
        country=country,
        is_primary=is_primary,
    )
    db.add(address)
    db.commit()
    db.refresh(address)
    return address
```

The status bar at the bottom indicates "Ln 29, Col 1" and "Python".

Функция (create_address_for_user) для работы с адресами:

- Принимает все необходимые параметры адреса (сессию БД, user_id, улицу, город, страну, штат, почтовый код, флаг is_primary)
- Создает новый адрес для пользователя с теми же параметрами
- Сохраняет адрес в базу данных
- Возвращает созданный объект адреса

Часть 2: Запрос связанных данных



The screenshot shows a code editor with two tabs open: `main.py` and `address.py`. The `main.py` tab is active and contains the following code:

```

File Edit Selection View Go Run Terminal Help ← → 🔍 lab2
EXPLORER main.py address.py
LAB2
  > .env
  > crud
    > __pycache__
      address.py
      user.py
  > db
    > __pycache__
      models
        > __pycache__
          __init__.py
          address.py
          user.py
          base.py
          session.py
    > migrations
    > .gitignore
    > python-version
    & alembic.ini
  docker-compose.yml
  main.py
  pyproject.toml
  README.md
  uv.lock

@ OUTLINE
@ TIMELINE
  ⌂ 0 △ 0
  In 31, Col 36  Spaces: 4  UTF-8  LF  {} Python  ⚙  ⌂
```

```

def main():
    db = SessionLocal()
    try:
        user = create_user(db, username="john doe", email="jdoe@example.com")
        print(f"created: {user.username}")

        address = create_address_for_user(
            db,
            user_id=user.id,
            street="21 Wall St",
            city="New York",
            state="New York",
            zip_code="10005",
            country="USA",
            is_primary=True,
        )
        print(f"created address for {user.username}: {address.street}, {address.city}")

        address = create_address_for_user(
            db,
            user_id=user.id,
            street="132 Central Prk W",
            city="New York",
            state="New York",
            zip_code="10023",
            country="USA",
            is_primary=False,
        )
        print(f"created address for {user.username}: {address.street}, {address.city}")

        stmt = select(User).options(selectinload(User.addresses))
        users = db.scalars(stmt).all()
    finally:
        pass

```

The code defines a `main()` function that creates a user with username "john doe" and email "jdoe@example.com". It then creates two addresses for this user: one at "21 Wall St" in New York City, and another at "132 Central Prk W" in New York City. Finally, it performs a query to select all users and their addresses.

- Создание сессии БД: `db = SessionLocal()`
- Добавление пользователя: `create_user(db, username=..., email=...)`
- Добавление двух адресов для предварительно созданного пользователя: `create_address_for_user(db, user_id=..., street=..., city=..., ...)`.
- После наполнения выполняем запрос на выборку всех пользователей с адресами (чтобы убедиться, что данные добавлены) и выводим их.

The screenshot shows a Microsoft Visual Studio Code window with the following details:

- File Explorer:** Shows a project structure with files like `main.py`, `address.py`, `base.py`, `session.py`, `migrations`, `.gitignore`, `.python-version`, `alembic.ini`, `docker-compose.yml`, `pyproject.toml`, `README.md`, and `uv.lock`.
- Terminal:** The terminal tab is active, showing command-line output from a Python application. The logs indicate the creation of a user named "John Doe" with ID 132, and subsequent database queries for users and addresses.
- Output:** Shows standard output and error logs.
- Debug Console:** Shows the current state of the application's memory.
- Problems:** Shows any errors or warnings found in the code.
- Ports:** Shows open ports on the system.

```
File Edit Selection View Go Run Terminal Help ← → 🔍 lab2 08 🌐 🔍 ... EXPLORE LAB2 .venv crud _pycache_ address.py user.py db _pycache_ models _pycache_ _init_.py address.py user.py base.py session.py migrations .gitignore .python-version alembic.ini docker-compose.yml main.py pyproject.toml README.md uv.lock 🔍 main.py x 🔍 main.py 50 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,015 INFO sqlalchemy.engine.Engine [cached since 0.02205s ago] {'pk_1': '6488822d-4846-4a99-9892-5b128545469f'} Created address for John Doe: 132 Central Prk W, New York 2025-10-20 01:36:07,021 INFO sqlalchemy.engine.Engine SELECT users.id, users.username, users.email, users.created_at, users.updated_at 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.02333s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'} 2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS users_created_at, users.updated_at FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.02333s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'} 2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS users_created_at, users.updated_at FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.02333s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'} 2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS users_created_at, users.updated_at FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.02333s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'} 2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS users_created_at, users.updated_at FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [cached since 0.02333s ago] {'pk_1': '8f7299ff-ec1f-49d2-a259-343d00c1de12'} 2025-10-20 01:36:07,013 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.created_at AS users_created_at, users.updated_at FROM users WHERE users.id = %(pk_1)s 2025-10-20 01:36:07,009 INFO sqlalchemy.engine.Engine [generated in 0.00071s] {} 2025-10-20 01:36:07,027 INFO sqlalchemy.engine.Engine SELECT addresses.user_id AS addresses_user_id, addresses.id AS addresses_id, addresses.street AS addresses_street, addresses.city AS addresses_city, addresses.state AS addresses_state, addresses.zip_code AS addresses_zip_code, addresses.country AS addresses_country, addresses.is_primary AS addresses_is_primary, addresses.created_at AS addresses_created_at, addresses.updated_at AS addresses_updated_at FROM addresses WHERE addresses.user_id IN (%(primary_keys_1)s) 2025-10-20 01:36:07,029 INFO sqlalchemy.engine.Engine [generated in 0.00165s] {'primary_keys_1': '6488822d-4846-4a99-9892-5b128545469f'} John Doe (jdoe@example.com) 21 Wall St, New York, USA 132 Central Prk W, New York, USA 2025-10-20 01:36:07,035 INFO sqlalchemy.engine.Engine ROLLBACK
```

Часть 3: Последующие работы с БД и миграции

1) Модификация модели User (models/user.py)

```
db > models > user.py
1  from sqlalchemy.orm import (
2      Mapped,
3      mapped_column,
4      relationship,
5  )
6  from uuid import uuid4
7  from datetime import datetime
8
9  from db.base import Base
10
11
12  class User(Base):
13      __tablename__ = 'users'
14
15      id: Mapped[str] = mapped_column(
16          primary_key=True,
17          default=lambda: str(uuid4())
18      )
19      username: Mapped[str] = mapped_column(nullable=False, unique=True)
20      email: Mapped[str] = mapped_column(nullable=False, unique=True)
21      description: Mapped[str] = mapped_column(default="", nullable=True)
22
23      created_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
24      updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
25
26      addresses = relationship("Address", back_populates="user")
27      orders = relationship("Order", back_populates="user")
28
```

Добавлено новое строковое поле description:

- description: Mapped[str] = mapped_column(default="", nullable=True)
 - Тип: Mapped[str]
 - Значение по умолчанию: пустая строка ("").
 - Разрешено значение NULL (nullable=True)

2) Создание модели Product (models/product.py)

```
db > models > product.py
1  from datetime import datetime
2  from sqlalchemy import Float
3  from sqlalchemy.orm import Mapped, mapped_column
4  from uuid import uuid4
5
6
7  from db.base import Base
8
9  class Product(Base):
10     __tablename__ = 'products'
11
12     id: Mapped[str] = mapped_column(primary_key=True, default=lambda: str(uuid4()))
13     name: Mapped[str] = mapped_column(nullable=False)
14     description: Mapped[str] = mapped_column(nullable=True)
15     price: Mapped[float] = mapped_column(nullable=False)
16
17     created_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
18     updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now, nullable=False)
19
```

Создана модель Product со следующими полями:

- id (первичный ключ, UUID)
 - name (название продукта, не NULL)
 - description (описание, может быть NULL)
 - price (цена, не NULL)
 - created_at и updated_at (даты создания и обновления)

3) Создание модели Order и ассоциативной таблицы (models/order.py)

The screenshot shows a Python application structure in the Explorer pane. The `db` directory contains `models`, `__pycache__`, and `base` sub-directories. `models` contains `__init__.py` and `address.py`. `base` contains `Base`, `order.py`, `product.py`, `user.py`, `base.py`, `session.py`, and `migrations`. `order.py` is currently selected and its content is displayed in the Editor pane.

```
from datetime import datetime
from sqlalchemy import ForeignKey, Table, Column
from sqlalchemy.orm import Mapped, mapped_column, relationship
from uuid import uuid4

from db.base import Base

order_product_association = Table(
    'order_products',
    Base.metadata,
    Column('order_id', ForeignKey('orders.id'), primary_key=True),
    Column('product_id', ForeignKey('products.id'), primary_key=True)
)

class Order(Base):
    __tablename__ = 'orders'

    id: Mapped[str] = mapped_column(primary_key=True, default=lambda: str(uuid4()))
    user_id: Mapped[str] = mapped_column(ForeignKey('users.id'), nullable=False)
    address_id: Mapped[str] = mapped_column(ForeignKey('addresses.id'), nullable=False)
    created_at: Mapped[datetime] = mapped_column(default=datetime.now, nullable=False)

    user = relationship("User", back_populates="orders")
    address = relationship("Address")
    products = relationship("Product", secondary=order_product_association)
```

The Terminal pane at the bottom shows the command `uvicorn` running, with output indicating the application is listening on port 0.0.0.0.

Модель Order: связывает пользователя (user_id), адрес доставки (address_id) и продукты через ассоциативную таблицу order_products.

Поля модели Order:

- id (первичный ключ, UUID)
 - user_id (внешний ключ на users.id)
 - address_id (внешний ключ на addresses.id)
 - created_at (дата создания заказа)

Ассоциативная таблица `order_products`: реализует связь «многие-ко-многим» между заказами и продуктами.

Связи между моделями

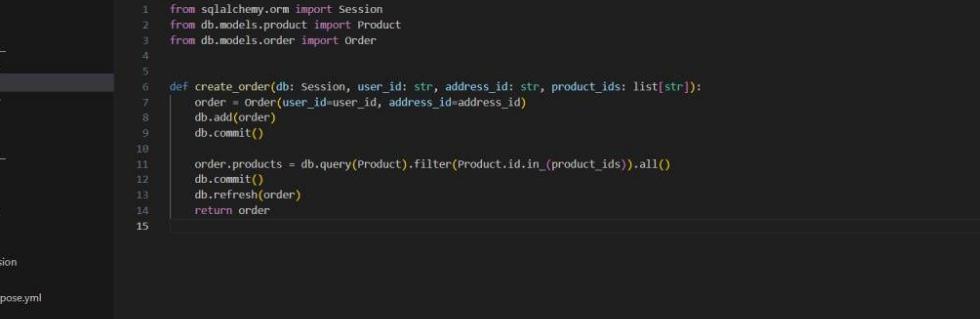
- User → Order: Один-ко-многим (users.orders).
 - Order → Product: Многие-ко-многим через order_products.
 - Order → Address: Многие-к-одному (один заказ привязан к одному адресу).

Файл product.py

```
product.py
crud > product.py
1  from sqlalchemy.orm import Session
2  from db.models.product import Product
3
4
5  def create_product(db: Session, name: str, description: str, price: float):
6      product = Product(name=name, description=description, price=price)
7      db.add(product)
8      db.commit()
9      db.refresh(product)
10
11  return product
```

- определена функция `create_product` для добавления новых продуктов в базу данных.

Файл order.py



The screenshot shows a code editor interface with the following details:

- File Explorer (Left):** Shows the project structure. The `LAB2` folder contains files like `crud.py`, `order.py`, `product.py`, `user.py`, and `db`. The `db` folder contains `base.py`, `models.py`, and `session.py`. There are also `migrations`, `.gitignore`, and configuration files (`.python-version`, `alembic.ini`, `docker-compose.yml`, `main.py`, `pyproject.toml`, `README.md`, and `uvlock`).
- Terminal (Top):** Shows the path `crud > order.py`.
- Code Editor (Center):** Displays the `order.py` file content. The code defines a function `create_order` that creates an `Order` object with `user_id` and `address_id`, adds it to a session, and commits the transaction. It then queries the database for products matching the provided `product_ids` and refreshes the `Order` object.
- Status Bar (Bottom):** Shows the current line (Ln 15), column (Col 1), spaces (Spaces 4), encoding (UTF-8), and language (Python).

- определена функция `create_order` для создания заказов.

Файл main.py – добавление тестовых данных

```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 lab2
EXPLORER LAB2
main.py
main.py
def main():
    from db.session import SessionLocal
    db = SessionLocal()
    try:
        user = create_user(db, username="John Doe", email="jdoe@example.com")
        user.description = "VIP клиент"
        db.commit()
        db.refresh(user)

        address = create_address_for_user(db, user_id=user.id,
                                           street="21 Wall st",
                                           city="New York",
                                           country="USA",
                                           state="NY",
                                           zip_code="10005",
                                           is_primary=True)
    finally:
        db.close()

    products = []
    for i in range(5):
        products.append(create_product(db, f"Product {i+1}", f"Description {i+1}", price=10.0*(i+1)))

    for i in range(5):
        create_order(db, user_id=user.id, address_id=address.id, product_ids=[p.id for p in products])
    if __name__ == "__main__":
        main()
```

Ln 25, Col 9 Spaces: 4 UTF-8 ⚙ Python ⚙

- Добавлено 5 записей в таблицу products
- Добавлено 5 заказов в таблицу orders

Каждый заказ привязан к существующему пользователю и адресу.
В ассоциативную таблицу добавлены связи между заказами и продуктами.

Задача: Проведение миграций

```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 lab2
EXPLORER LAB2
migrations
migrations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + ⚙ 🌐 ... | 🔍 ×

● (lab2) PS C:\Users\Anastasia\Desktop\lab2> alembic revision --autogenerate -m "add_orders, products, change_user"
INFO [alembic.runtime.migration] Context impl PostgresImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'products'
INFO [alembic.autogenerate.compare] Detected added table 'orders'
INFO [alembic.autogenerate.compare] Detected added table 'order_products'
INFO [alembic.autogenerate.compare] Detected added column 'users.description'
Generating C:/Users/Anastasia/Desktop/lab2/migrations/versions/8efdf2ac7cf0.add_orders_products_change_user.py ...
done
● (lab2) PS C:\Users\Anastasia\Desktop\lab2> alembic upgrade head
INFO [alembic.runtime.migration] Context impl PostgresImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] running upgrade 5fb292750b0b -> 8efdf2ac7cf0, add_orders, products, change_user
● (lab2) PS C:\Users\Anastasia\Desktop\lab2>
```

Создание миграций:

alembic revision --autogenerate -m "add orders, products, change user"

- обнаружено добавление таблиц products, orders и order_products.
- обнаружено добавленное поле users.description

Применение миграции к БД:

alembic upgrade head

```
(lab2) PS C:\Users\Anastasia\Desktop\lab2> python ./main.py
2025-10-20 02:13:10,754 INFO sqlalchemy.engine.Engine SELECT orders.id AS orders_id, orders.user_id AS orders_user_id, orders.address_id AS orders_address_id, orders.created_at AS orders_created_at
FROM orders
WHERE orders.id = %(pk_1)s
2025-10-20 02:13:10,757 INFO sqlalchemy.engine.Engine [cached since 0.2932s ago] {'pk_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3'}
2025-10-20 02:13:10,768 INFO sqlalchemy.engine.Engine SELECT products.id AS products_id, products.name AS products_name, products.description AS products_description, products.price AS products_price, products.created_at AS products_created_at, products.updated_at AS products_updated_at
FROM products, order_products
WHERE %(param_1)s = order_products.order_id AND products.id = order_products.product_id
2025-10-20 02:13:10,761 INFO sqlalchemy.engine.Engine [cached since 0.2933s ago] {'param_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3'}
2025-10-20 02:13:10,764 INFO sqlalchemy.engine.Engine INSERT INTO order_products (order_id, product_id) VALUES (%(order_id_0)s, %(product_id_0)s), %(order_id_1)s, %(product_id_1)s, %(order_id_2)s, %(product_id_2)s), %(order_id_3)s, %(product_id_3)s), %(order_id_4)s, %(product_id_4)s)
2025-10-20 02:13:10,765 INFO sqlalchemy.engine.Engine [cached since 0.2988s ago] {insertmanyvalues: 1/1 (unordered)} {'order_id_0': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_0': '962be1a6-599c-aec9-b1c5c92e982e', 'order_id_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_1': '011d4d24-7018-4ddc-8bc5-3bf11daef35f9', 'order_id_2': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_2': '8515188bd-91a7-4910-98d0-7edeb0873a81', 'order_id_3': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_3': '2fbfc195e-0f63-476a-ba68-63ee2e552bf3', 'order_id_4': '3a9223fa-386b-436d-9c6e-4fb5af4605f3', 'product_id_4': 'ab4f8396-9c9d-4c02-a749-b31e135a5181'}
2025-10-20 02:13:10,767 INFO sqlalchemy.engine.Engine COMMIT
2025-10-20 02:13:10,771 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-20 02:13:10,772 INFO sqlalchemy.engine.Engine SELECT orders.id, orders.user_id, orders.address_id, orders.created_at
FROM orders
WHERE orders.id = %(pk_1)s
2025-10-20 02:13:10,773 INFO sqlalchemy.engine.Engine [cached since 0.29s ago] {'pk_1': '3a9223fa-386b-436d-9c6e-4fb5af4605f3'}
2025-10-20 02:13:10,775 INFO sqlalchemy.engine.Engine ROLLBACK
(lab2) PS C:\Users\Anastasia\Desktop\lab2>
```

	id	user_id	address_id	created_at
1	5f52717d-efe1-4dbb-9cc4-1160fe31...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13bbfaf1...	2025-10-20 02:13:10.447132
2	0eaea4ae-1cf4-4372-8401-be494c46...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13bbfaf1...	2025-10-20 02:13:10.562237
3	ab6f9c91-13f9-4432-b97d-a88db1ef...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13bbfaf1...	2025-10-20 02:13:10.610574
4	5046f7b7-f179-4a2e-a0f1-b289ee98e...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13bbfaf1...	2025-10-20 02:13:10.736588
5	3a9223fa-386b-436d-9c6e-4fb5af460...	6488822d-4846-4a99-9892-5b1285454...	38f5dd3e-afa2-42f4-8449-e13bbfaf1...	2025-10-20 02:13:10.736588

The screenshot shows the pgAdmin 4 interface with a database connection named 'my_db/user@docker-db'. A SQL query is run:

```
1 SELECT * FROM my_db.public.products;
```

The resulting data table is:

	[PK] character varying	name character varying	description character varying	price double precision	created_at timestamp without time zone	updated_at timestamp without time zone
1	962be1a6-509c-4372-aec9-b1c5c92e98...	Product 1	Description 1	10	2025-10-20 02:13:10.364749	2025-10-20 02:13:10.364753
2	011d4242-7018-40de-8bc5-3bf11da635...	Product 2	Description 2	20	2025-10-20 02:13:10.381351	2025-10-20 02:13:10.381354
3	855188bd-91a7-4910-98d0-7dede0673...	Product 3	Description 3	30	2025-10-20 02:13:10.392197	2025-10-20 02:13:10.392201
4	2bbc195e-0f63-476a-ba68-63e2e552bff3	Product 4	Description 4	40	2025-10-20 02:13:10.402294	2025-10-20 02:13:10.402297
5	ab4f8390-9c9d-4c02-a749-b31e135a51...	Product 5	Description 5	50	2025-10-20 02:13:10.413675	2025-10-20 02:13:10.413678

The screenshot shows the pgAdmin 4 interface with a database connection named 'my_db/user@docker-db'. A SQL query is run:

```
1 SELECT * FROM my_db.public.users;
```

The resulting data table is:

	[PK] character varying	username character varying	email character varying	created_at timestamp without time zone	updated_at timestamp without time zone	description character varying
1	6488822d-4846-4a99-9892-5b1285454...	John Doe	jdoe@example.co...	2025-10-20 02:13:10.329225	2025-10-20 02:13:10.329223	VIP клиент

The screenshot shows the pgAdmin 4 interface. At the top, there's a browser-like header with tabs for 'pgAdmin 4' and 'Новая вкладка'. Below it is a toolbar with various icons. The main window has a title bar 'Добро пожаловать x my_db/user@docker-db*'. It contains a 'Запрос' (Query) tab with the SQL command: 'SELECT * FROM my_db.public.addresses;'. To the right of the query is a 'Scratch Pad' tab. Below the query area is a 'Data Output' tab showing the results of the query:

	<code>[PK] character varying</code>	<code>user_id</code> <code>character varying</code>	<code>street</code> <code>character varying</code>	<code>city</code> <code>character varying</code>	<code>state</code> <code>character varying</code>	<code>zip_code</code> <code>character varying</code>	<code>country</code> <code>character varying</code>	<code>is_primary</code> <code>boolean</code>	<code>created_at</code> <code>timestamp with time zone</code>
1	8f7299ff-ec1f-49d2-a259-343d00c1de12	6488822d-4846-4a99-9892-5b128545469f	132 Central Prk W	New York	New York	10023	USA	false	2025-10-20
2	38f5dd3e-afa2-42f4-8449-e13fbfafe1ea	6488822d-4846-4a99-9892-5b128545469f	21 Wall St	New York	NY	10005	USA	true	2025-10-20

At the bottom of the pgAdmin window, there's a status bar with 'Total rows: 2' and 'Query complete 00:00:00.101'.

Ответы на вопросы

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

SQLAlchemy предлагает два основных подхода к маппингу:

Декларативный маппинг (использован в лабораторной работе) - современный подход, где модели определяются как классы с аннотациями типов. Позволяет компактно описывать структуру таблиц непосредственно в классах Python с использованием `Mapped` и `mapped_column`.

Императивный (классический) маппинг - более традиционный подход, где таблицы и классы определяются отдельно, затем связываются через `mapper`. Требует больше кода, но дает полный контроль над процессом маппинга.

Когда использовать:

- Декларативный – более современный подход, подходит для новых проектов, когда нужна быстрота разработки (быстро описать модели с минимальным кодом) и читаемость кода (аннотации типов + минимизация boilerplate-кода)
- Императивный - при работе с унаследованными схемами (когда БД уже существует, и ее структура фиксирована; когда нужно маппить существующие сложные таблицы; требуется точное соответствие с существующей схемой) или когда требуется тонкая настройка маппинга, нестандартные типы данных и кастомные преобразования

2. Как Alembic отслеживает текущую версию базы данных?

Alembic хранит текущую версию в специальной таблице (по умолчанию alembic_version) в самой базе данных. В таблице лежит version_num — идентификатор последней применённой миграции.

При выполнении команд alembic upgrade или alembic downgrade система:

- Читает текущую версию из таблицы alembic_version
- Определяет какие миграции нужно применить для достижения целевого состояния
- Последовательно выполняет миграционные скрипты
- Обновляет запись в таблице alembic_version

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

В работе реализованы 3 типа реляционных связей:

- Один-ко-многим (One-to-Many) - связь между пользователем и заказами, где один пользователь может иметь несколько заказов, но каждый заказ принадлежит только одному пользователю.
- Многие-к-одному (Many-to-One) - обратная сторона предыдущей связи, а также связь между заказом и адресом доставки, где несколько заказов могут быть привязаны к одному адресу.
- Многие-ко-многим (Many-to-Many) - реализована через ассоциативную таблицу order_products для связи заказов и продуктов. Один заказ может содержать несколько продуктов, и один продукт может присутствовать в нескольких заказах.

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

Миграция БД — это управляемый процесс изменения схемы базы данных, который включает модификацию таблиц, столбцов, индексов и связей между таблицами.

Важность миграций обусловлена:

- Безопасностью изменений - миграции позволяют применять изменения к структуре БД без потери данных и с возможностью отката
- Версионным контролем - каждая миграция имеет уникальный идентификатор, что позволяет точно отслеживать историю изменений схемы БД
- Совместной разработкой - команды разработчиков могут работать независимо, создавая миграции, которые затем объединяются
- Воспроизводимостью - одинаковое состояние БД может быть достигнуто на разных окружениях (разработка, тестирование, продакшен)

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

В SQLAlchemy отношения многие-ко-многим реализуются через ассоциативные таблицы:

- Создание ассоциативной таблицы - промежуточной таблицы, содержащей только

внешние ключи на связываемые таблицы.

- Настройка relationship - в основной модели указывается связь через параметр secondary

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

- 1) **Анализ ситуации** - определить какие именно миграции конфликтуют и в чем заключается конфликт (изменение одних и тех же таблиц, столбцов и т.д.)
- 2) **Создание merge-миграции** - выполнить команду alembic merge для создания миграции, объединяющей изменения из конфликтующих версий
- 3) **Ручное разрешение конфликтов** - если автоматическое объединение невозможно, отредактировать миграционные скрипты вручную, устранив противоречия
- 4) **Тестирование** - проверить корректность merge-миграции на тестовой базе данных перед применением к основной
- 5) **Коммуникация в команде** - уведомить других разработчиков о разрешении конфликта и предоставить обновленные миграции
- 6) **Профилактика** - установить правила работы в команде, минимизирующие вероятность конфликтов (например, согласование изменений схемы БД перед созданием миграций)

Вывод:

В ходе выполнения лабораторной работы были освоены практические навыки работы с SQLAlchemy ORM и системой миграций Alembic, в том числе:

- создание и модификация ORM-моделей (описание структуры БД на уровне Python-классов, добавление новых полей (как поле description для пользователя) и настройка связей между таблицами).
- работа со связями через ассоциативные таблицы (реализована связь один-ко-многим (пользователь-заказы) и многие-ко-многим (заказы-продукты))
- работа с системой миграций (создание миграционных скриптов, отслеживание изменений в моделях и применение их к БД)

Работа показала, что использование ORM и системы миграций значительно упрощает разработку и сопровождение приложений с базами данных. Миграции позволяют безопасно изменять структуру БД при активной разработке, а SQLAlchemy избавляет от необходимости писать сложные SQL-запросы вручную.