Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий – РТФ

Основы работы с Docker и PostgreSQL

Лабораторное задание №2

Группа: РИМ-150950
Студент: Владимиров Н.А.

Преподаватель: Кузьмин Д. И.

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:

- Инициализация БД и создание ОРМ для пользователя и адреса.
- Инициализация и миграция метаданных.
- Выполнить запрос связанных данных

Ход работы:

1. Установка библиотек

- sqlalchemy - ORM для работы с базами данных

- alembic - система миграций базы данных

- psycopg2-binary - драйвер для PostgreSQL (для будущего использования)

- asyncpg - асинхронный драйвер для PostgreSQL

2. Создание ORM моделей

2.1 Модель User (Пользователь)

- id - UUID, первичный ключ

- username - уникальное имя пользователя

- email - уникальный email

- description - дополнительное описание (добавлено позже)

- created_at, updated_at - временные метки

```python
class User(Base):
    __tablename__ = 'users'

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid4)
    username: Mapped[str] = mapped_column(nullable=False, unique=True)
    email: Mapped[str] = mapped_column(nullable=False, unique=True)
    description: Mapped[str] = mapped_column()
    created_at: Mapped[datetime] = mapped_column(default=datetime.now)
    updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now)
    addresses = relationship("Address", back_populates="user")
    orders = relationship("Order", back_populates="user")
```

2.2 Модель Address (Адрес)

- id - UUID, первичный ключ

- user_id - внешний ключ на User

- street, city, state, zip_code, country - адресные данные

- is_primary - флаг основного адреса

- created_at, updated_at - временные метки

```python
class Address(Base):
    __tablename__ = 'addresses'

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid4)
    user_id: Mapped[UUID] = mapped_column(ForeignKey('users.id'), nullable=False)
    street: Mapped[str] = mapped_column(nullable=False)
    city: Mapped[str] = mapped_column(nullable=False)
    state: Mapped[str] = mapped_column()
    zip_code: Mapped[str] = mapped_column()
    country: Mapped[str] = mapped_column(nullable=False)
    is_primary: Mapped[bool] = mapped_column(default=False)
    created_at: Mapped[datetime] = mapped_column(default=datetime.now)
    updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now)
    user = relationship("User", back_populates="addresses")
    orders = relationship("Order", back_populates="delivery_address")
```

2.3 Модель Product (Продукт)

- id - UUID, первичный ключ

- name - название продукта

- description - описание

- price - цена (Decimal)

- created_at, updated_at - временные метки

```python
class Product(Base):
    __tablename__ = 'products'

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid4)
    name: Mapped[str] = mapped_column(nullable=False)
    description: Mapped[str] = mapped_column()
    price: Mapped[Decimal] = mapped_column(nullable=False)
    created_at: Mapped[datetime] = mapped_column(default=datetime.now)
    updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now)
    orders = relationship("Order", secondary=order_product_association, back_populates="products")
```

2.4 Модель Order (Заказ)

- id - UUID, первичный ключ

- user_id - внешний ключ на User

- delivery_address_id - внешний ключ на Address

- total_amount - общая сумма заказа

- status - статус заказа (pending, processing, shipped, delivered)

- created_at, updated_at - временные метки

```python
class Order(Base):
    __tablename__ = 'orders'

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid4)
    user_id: Mapped[UUID] = mapped_column(ForeignKey('users.id'), nullable=False)
    delivery_address_id: Mapped[UUID] = mapped_column(ForeignKey('addresses.id'), nullable=False)
    total_amount: Mapped[Decimal] = mapped_column(nullable=False)
    status: Mapped[str] = mapped_column(default="pending")
    created_at: Mapped[datetime] = mapped_column(default=datetime.now)
    updated_at: Mapped[datetime] = mapped_column(default=datetime.now, onupdate=datetime.now)
    user = relationship("User", back_populates="orders")
    delivery_address = relationship("Address", back_populates="orders")
    products = relationship("Product", secondary=order_product_association, back_populates="orders")
```

3. Типы связей между таблицами

3.1 Один-ко-многим (One-to-Many):

- User → Address (один пользователь, много адресов)

- User → Order (один пользователь, много заказов)

- Address → Order (один адрес, много заказов)

3.2 Многие-к-одному (Many-to-One):

- Address → User (много адресов, один пользователь)

- Order → User (много заказов, один пользователь)

- Order → Address (много заказов, один адрес доставки)

3.3 Многие-ко-многим (Many-to-Many):

- Order ↔ Product (через промежуточную таблицу order_product)

4. Инициализация и настройка Alembic

4.1 Инициализирована папка миграций: alembic init migrations

4.2 Настроен alembic.ini с строкой подключения к БД

```
[alembic]

script_location = %(here)s/migrations
prepend_sys_path = .
path_separator = os
sqlalchemy.url = driver://user:pass@localhost/dbname
```

4.3 Настроен migrations/env.py для использования Base.metadata

```
config = context.config

if config.config_file_name is not None:
    fileConfig(config.config_file_name)

from models import Base
target_metadata = Base.metadata
```

5. Создание и применение миграций

5.1 Первая миграция: Создание таблиц users и addresses

```
engine = create_engine(
    DATABASE_URL,
    echo=True
)

session_factory = sessionmaker[Session](bind=engine)

with session_factory() as session:

    user1 = User(username="john_doe", email="john.doe@example.com")
    session.add(user1)
    session.flush()
    address1_1 = Address(
        user_id=user1.id,
        street="123 Main Street",
        city="New York",
        state="NY",
        zip_code="10001",
        country="USA",
        is_primary=True
    )
    address1_2 = Address(
        user_id=user1.id,
        street="456 Park Avenue",
        city="New York",
        state="NY",
        zip_code="10022",
        country="USA",
        is_primary=False
    )
    session.add(address1_1)
    session.add(address1_2)
```

5.2 Вторая миграция: Добавление поля description к User, создание таблиц products, orders и order_product

```
stmt = select(User).options(selectinload(User.addresses))
users = session.execute(stmt).scalars().all()

order1 = Order(
    user_id=users[0].id,
    delivery_address_id=users[0].addresses[0].id,
    total_amount=Decimal("2299.98"),
    status="processing"
)
order1.products = [products[0], products[1]]
session.add(order1)
```

6. Наполнение базы данных

- 5 пользователей с их адресами (через populate_db.py)

```
user1 = User(username="john_doe", email="john.doe@example.com")
session.add(user1)
session.flush()
address1_1 = Address(
    user_id=user1.id,
    street="123 Main Street",
    city="New York",
    state="NY",
    zip_code="10001",
    country="USA",
    is_primary=True
)
address1_2 = Address(
    user_id=user1.id,
    street="456 Park Avenue",
    city="New York",
    state="NY",
    zip_code="10022",
    country="USA",
    is_primary=False
)
session.add(address1_1)
session.add(address1_2)
```

- 5 продуктов (ноутбук, смартфон, наушники, планшет, умные часы)

```python
with session_factory() as session:
    products = [
        Product(
            name="Ноутбук Dell XPS 15",
            description="Мощный ноутбук для работы и творчества",
            price=Decimal("1299.99")
        ),
        Product(
            name="Смартфон iPhone 15 Pro",
            description="Флагманский смартфон с продвинутой камерой",
            price=Decimal("999.99")
        ),
        Product(
            name="Наушники Sony WH-1000XM5",
            description="Беспроводные наушники с активным шумоподавлением",
            price=Decimal("399.99")
        ),
        Product(
            name="Планшет iPad Pro 12.9",
            description="Профессиональный планшет для дизайна и работы",
            price=Decimal("1099.99")
        ),
        Product(
            name="Умные часы Apple Watch Series 9",
            description="Фитнес-трекер и умные часы премиум класса",
            price=Decimal("399.99")
        )
    ]
```

- 5 заказов с различными комбинациями продуктов

```python
order1 = Order(
    user_id=users[0].id,
    delivery_address_id=users[0].addresses[0].id,
    total_amount=Decimal("2299.98"),
    status="processing"
)
order1.products = [products[0], products[1]]
session.add(order1)

order2 = Order(
    user_id=users[1].id,
    delivery_address_id=users[1].addresses[0].id,
    total_amount=Decimal("399.99"),
    status="shipped"
)
order2.products = [products[2]]
session.add(order2)

order3 = Order(
    user_id=users[2].id,
    delivery_address_id=users[2].addresses[0].id,
    total_amount=Decimal("799.98"),
    status="pending"
)
order3.products = [products[2], products[4]]
session.add(order3)

order4 = Order(
    user_id=users[3].id,
    delivery_address_id=users[3].addresses[0].id,
    total_amount=Decimal("1099.99"),
    status="delivered"
)
```

```python
order4 = Order(
    user_id=users[3].id,
    delivery_address_id=users[3].addresses[0].id,
    total_amount=Decimal("1099.99"),
    status="delivered"
)
order4.products = [products[3]]
session.add(order4)


order5 = Order(
    user_id=users[4].id,
    delivery_address_id=users[4].addresses[0].id,
    total_amount=Decimal("2799.97"),
    status="processing"
)
order5.products = [products[0], products[2], products[3]]
session.add(order5)
```

7. Запрос связанных данных

Реализован запрос с использованием selectinload для эффективной загрузки связанных данных:

```
with session_factory() as session:
    stmt = select(User).options(selectinload(User.addresses))
    users = session.execute(stmt).scalars().all()
```

Вывод: В ходе данной лабораторной работы были освоины принципы работы с библиотеками SQLAlchemy и Alembic для создания и управления реляционными базами данных на Python, изучены механизмы миграции базы данных. Были использованы:

SQLAlchemy 2.0 с современным синтаксисом Mapped[] и mapped_column()

- Alembic для управления миграциями

- PostgreSQL для демонстрации

- Связи: один-ко-многим, многие-к-одному, многие-ко-многим

- Автоматическая генерация миграций через --autogenerate

Ответы на вопросы:

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

1.1 Declarative Mapping (Декларативный маппинг)

Для большинства проектов, особенно новых. Простой, читаемый и современный подход.

1.2 Imperative Mapping (Императивный маппинг)

Когда нужно маппить классы из сторонних библиотек или legacy-код

1.3 Classical Mapping (Классический маппинг)

Только для поддержки старых проектов. Не рекомендуется для новых проектов.

2. Как Alembic отслеживает текущую версию базы данных?

Alembic отслеживает текущую версию базы данных через специальную таблицу alembic_version, которая создается автоматически при первой миграции. Содержит одну строку с полем version_num, которое хранит идентификатор (revision ID) последней примененной миграции.

Механизм отслеживания:

- При выполнении alembic upgrade head Alembic читает текущую версию из таблицы alembic_version, сравнивает с доступными миграциями в папке migrations/versions/ и применяет все миграции, которые еще не были применены.

Проверка версии:

- alembic current показывает текущую версию

- alembic history показывает все доступные миграции

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

3.1 Один-ко-многим

- User → Address: Один пользователь может иметь много адресов

- User → Order: Один пользователь может иметь много заказов

- Address → Order: Один адрес может использоваться в нескольких заказах (как адрес доставки)

3.2 Многие-к-одному

- Address → User: Много адресов принадлежат одному пользователю

- Order → User: Много заказов принадлежат одному пользователю

- Order → Address: Много заказов могут использовать один адрес доставки

3.3 Многие-ко-многим

- Order ↔ Product: Один заказ может содержать несколько продуктов, и один продукт может быть в нескольких заказах

- Реализовано через промежуточную таблицу order_product с дополнительным полем quantity

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

Миграция базы данных - это процесс изменения структуры базы данных (схемы) с сохранением данных и возможностью отката изменений.

Миграции важны, так как позволяют отслеживать все изменения структуры БД во времени, безопасно изменять структуру БД, возвращаться к предыдущей версии схемы, а также обеспечивают одинаковую структуру БД на всех окружениях (dev, staging, production) и позволяют нескольким разработчикам работать над проектом, применяя одинаковые миграции.

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

Отношения многие-ко-многим обрабатываются через промежуточную (ассоциативную) таблицу.

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

1. Обнаружение конфликта

2. Проверка текущего состояния

3. Объединение веток

4. Применение merge-миграции

Либо ручное исправление down_revision в одной из конфликтующих миграций