

Лабораторная работа 2

Цель работы:

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

Ход работы:

```
PS C:\lab2_final> dir
PS C:\lab2_final> @"
>> from sqlalchemy import Column, String, Boolean, ForeignKey, DateTime
>> from sqlalchemy.orm import DeclarativeBase
>> import uuid
>> from datetime import datetime
>>
>> class Base(DeclarativeBase):
>>     pass
>>
>> class User(Base):
>>     __tablename__ = 'users'
>>
>>     id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
>>     username = Column(String(50), nullable=False, unique=True)
>>     email = Column(String(100), nullable=False, unique=True)
>>     created_at = Column(DateTime, default=datetime.now)
>>     updated_at = Column(DateTime, default=datetime.now, onupdate=datetime.now)
>>
>> class Address(Base):
>>     __tablename__ = 'addresses'
>>
>>     id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
>>     user_id = Column(String(36), ForeignKey('users.id'), nullable=False)
>>     street = Column(String(200), nullable=False)
>>     city = Column(String(100), nullable=False)
>>     state = Column(String(100))
>>     zip_code = Column(String(20))
>>     country = Column(String(100), nullable=False)
>>     is_primary = Column(Boolean, default=False)
>>     created_at = Column(DateTime, default=datetime.now)
>>     updated_at = Column(DateTime, default=datetime.now, onupdate=datetime.now)
>>     "@ | Out-File -FilePath database.py -Encoding utf8
PS C:\lab2_final> alembic init migrations
```

Скриншот 1. Создание ОРМ для пользователя и данных

```
>>     id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
>>     username = Column(String(50), nullable=False, unique=True)
>>     email = Column(String(100), nullable=False, unique=True)
>>     created_at = Column(DateTime, default=datetime.now)
>>     updated_at = Column(DateTime, default=datetime.now, onupdate=datetime.now)
>>
>> class Address(Base):
>>     __tablename__ = 'addresses'
>>
>>     id = Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
>>     user_id = Column(String(36), ForeignKey('users.id'), nullable=False)
>>     street = Column(String(200), nullable=False)
>>     city = Column(String(100), nullable=False)
>>     state = Column(String(100))
>>     zip_code = Column(String(20))
>>     country = Column(String(100), nullable=False)
>>     is_primary = Column(Boolean, default=False)
>>     created_at = Column(DateTime, default=datetime.now)
>>     updated_at = Column(DateTime, default=datetime.now, onupdate=datetime.now)
>>     "@ | Out-File -FilePath database.py -Encoding utf8
PS C:\lab2_final> alembic init migrations
Creating directory C:\lab2_final\migrations ... done
Creating directory C:\lab2_final\migrations\versions ... done
Generating C:\lab2_final\alembic.ini ... done
Generating C:\lab2_final\migrations\env.py ... done
Generating C:\lab2_final\migrations\README ... done
Generating C:\lab2_final\migrations\script.py.mako ... done
Please edit configuration/connection/logging settings in C:\lab2_final\alembic.ini before proceeding.
PS C:\lab2_final> Get-Content alembic.ini
# A generic, single database configuration.

[alembic]
# path to migration scripts.
# this is typically a path given in POSIX (e.g. forward slashes)
# format, relative to the token %(here)s which refers to the location of this
```

Скриншот 2.Инициализация миграций: использование alembic init migrations

```
sqlalchemy.url = sqlite:///lab2.db

PS C:\lab2_final> @"
>> from logging.config import fileConfig
>> from sqlalchemy import engine_from_config
>> from sqlalchemy import pool
>> from alembic import context
>> import os
>> import sys
>>
>> sys.path.append(os.getcwd())
>>
>> from database import Base
>>
>> config = context.config
>>
>> if config.config_file_name is not None:
>>     fileConfig(config.config_file_name)
>>
>> target_metadata = Base.metadata
>>
>> def run_migrations_offline():
>>     url = config.get_main_option('sqlalchemy.url')
>>     context.configure(
>>         url=url,
>>         target_metadata=target_metadata,
>>         literal_binds=True,
>>         dialect_opts={'paramstyle': 'named'},
>>     )
>>     with context.begin_transaction():
>>         context.run_migrations()
>>
>> def run_migrations_online():
>>     connectable = engine_from_config(
>>         config.get_section(config.config_ini_section, {}),
>>         prefix='sqlalchemy.',
>>         poolclass=pool.NullPool,
>>     )
>>     with connectable.connect() as connection:
>>         context.configure(
>>             connection=connection,
>>             target_metadata=target_metadata
>>         )
>>         with context.begin_transaction():
>>             context.run_migrations()
>>
>> if context.is_offline_mode():
>>     run_migrations_offline()
>> else:
>>     run_migrations_online()
>> "@ | Out-File -FilePath migrations\env.py -Encoding utf8 -Force
PS C:\lab2_final> dir
```

Скриншот 3. В migrations/[env.py](#) подставляем метаданные для миграции

```
PS C:\lab2_final> @'
>> """init
>>           es()
>> Revision ID: 001 БД:', tables
>> Revises:
>> Create Date: 2024-11-30 01:10:00
>>
>> """
>> from alembic import op
>> import sqlalchemy as sa
>>
>>
>> # revision identifiers, used by Alembic.
>> revision = '001'
>> down_revision = None
>> branch_labels = None
>> depends_on = None
>>
>>
>> def upgrade():
>>     # ### commands auto generated by Alembic - please adjust! ###
>>     op.create_table('users',
>>         sa.Column('id', sa.String(length=36), nullable=False),
>>         sa.Column('username', sa.String(length=50), nullable=False),
>>         sa.Column('email', sa.String(length=100), nullable=False),
>>         sa.Column('created_at', sa.DateTime(), nullable=True),
>>         sa.Column('updated_at', sa.DateTime(), nullable=True),
>>         sa.PrimaryKeyConstraint('id'),
>>         sa.UniqueConstraint('username'),
>>         sa.UniqueConstraint('email')
>>     )
>>     op.create_table('addresses',
>>         sa.Column('id', sa.String(length=36), nullable=False),
>>         sa.Column('user_id', sa.String(length=36), nullable=False),
>>         sa.Column('street', sa.String(length=200), nullable=False),
>>         sa.Column('city', sa.String(length=100), nullable=False),
>>         sa.Column('state', sa.String(length=100), nullable=True),
>>         sa.Column('zip_code', sa.String(length=20), nullable=True),
>>         sa.Column('country', sa.String(length=100), nullable=False),
>>         sa.Column('is_primary', sa.Boolean(), nullable=True),
>>         sa.Column('created_at', sa.DateTime(), nullable=True),
>>         sa.Column('updated_at', sa.DateTime(), nullable=True),
>>         sa.ForeignKeyConstraint(['user_id'], ['users.id'], ),
>>         sa.PrimaryKeyConstraint('id')
>>     )
>>     # ### end Alembic commands #####
>>
>>
>> def downgrade():
>>     # ### commands auto generated by Alembic - please adjust! ###
>>     op.drop_table('addresses')
>>     op.drop_table('users')
>>     # ### end Alembic commands #####
>>     '@ | Out-File -FilePath "migrations\versions\001_initial.py" -Encoding utf8
PS C:\lab2_final> Get-Content "migrations\versions\001_initial.py" | Select-Object -First 5
```

Скриншот 4. Создание файла миграции

```
Windows PowerShell      X + ▾

>> # ### commands auto generated by Alembic - please adjust! ###
>> op.drop_table('addresses')
>> op.drop_table('users')
>> # ### end Alembic commands ###
>> '@ | Out-File -FilePath "migrations\versions\001_initial.py" -Encoding utf8
PS C:\lab2_final> Get-Content "migrations\versions\001_initial.py" | Select-Object -First 5
"""init

Revision ID: 001
Revises:
Create Date: 2024-11-30 01:10:00
PS C:\lab2_final> alembic upgrade head
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 001, init
PS C:\lab2_final> python -c "
>> from sqlalchemy import create_engine, inspect
>> engine = create_engine('sqlite:///Lab2.db')
>> inspector = inspect(engine)
>> tables = inspector.get_table_names()
>> print('✖ Таблицы в БД:', tables)
>>
>> if 'users' in tables and 'addresses' in tables:
>>     print('✅ ВСЕ ТАБЛИЦЫ СОЗДАНЫ УСПЕШНО!')
>>     print('└─ Структура таблиц:')
>>     for table in ['users', 'addresses']:
>>         print(f'\n{table}:')
>>         for column in inspector.get_columns(table):
>>             print(f'    - {column['name']}: {column['type']}')
>> else:
>>     print('✖ Проблема с созданием таблиц')
>>
✖ Таблицы в БД: ['addresses', 'alembic_version', 'users']
✅ ВСЕ ТАБЛИЦЫ СОЗДАНЫ УСПЕШНО!

└─ Структура таблиц:

users:
- id: VARCHAR(36)
- username: VARCHAR(50)
- email: VARCHAR(100)
- created_at: DATETIME
- updated_at: DATETIME

addresses:
- id: VARCHAR(36)
- user_id: VARCHAR(36)
- street: VARCHAR(200)
- city: VARCHAR(100)
- state: VARCHAR(100)
- zip_code: VARCHAR(20)
- country: VARCHAR(100)
- is_primary: BOOLEAN
- created_at: DATETIME
- updated_at: DATETIME
```

Скриншот 5. Применение к базе данных последней миграции

```
-rcca-52c0e254d835', 'alice_brown', 'alice@example.com', '2025-11-30 01:45:24.761180', '2025-11-30 01:45:24.761181', 'aa787f1d-a530-4  
33e-b8f9-c445c48a6ba', 'charlie_davis', 'charlie@example.com', '2025-11-30 01:45:24.761196', '2025-11-30 01:45:24.761198')  
2025-11-30 01:45:24,780 INFO sqlalchemy.engine.Engine COMMIT  
 Пользователи созданы  
2025-11-30 01:45:24,796 INFO sqlalchemy.engine.Engine BEGIN (implicit)  
2025-11-30 01:45:24,808 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u  
ser_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at  
FROM users  
WHERE users.id = ?  
2025-11-30 01:45:24,809 INFO sqlalchemy.engine.Engine [generated in 0.00011s] ('b2d42a48-3fdd-4b6c-8bd2-f78c04f0dbca',)  
2025-11-30 01:45:24,823 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_  
primary, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  
2025-11-30 01:45:24,823 INFO sqlalchemy.engine.Engine [generated in 0.000153s] ('2f419758-4cb7-4a29-a8ef-0d65b0a68fc2', 'b2d42a48-3fdd  
-4b6c-8bd2-f78c04f0dbca', '123 Main St', 'New York', None, None, 'USA', 1, '2025-11-30 01:45:24.822567', '2025-11-30 01:45:24.822576'  
)  
2025-11-30 01:45:24,832 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u  
ser_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at  
FROM users  
WHERE users.id = ?  
2025-11-30 01:45:24,832 INFO sqlalchemy.engine.Engine [cached since 0.02418s ago] ('e0214ffc-3b3d-4033-b1c2-4bf0bf3d39e6',)  
2025-11-30 01:45:24,835 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_  
primary, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  
2025-11-30 01:45:24,838 INFO sqlalchemy.engine.Engine [cached since 0.01602s ago] ('1aa4ca0c-f266-42ef-9069-9d2fb5066012', 'e0214ffc-  
3b3d-4033-b1c2-4bf0bf3d39e6', '456 Oak Ave', 'Los Angeles', None, None, 'USA', 1, '2025-11-30 01:45:24.835201', '2025-11-30 01:45:24.  
835207')  
2025-11-30 01:45:24,839 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u  
ser_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at  
FROM users  
WHERE users.id = ?  
2025-11-30 01:45:24,839 INFO sqlalchemy.engine.Engine [cached since 0.03143s ago] ('a3141812-e33c-4641-b571-5f272673c22f',)  
2025-11-30 01:45:24,842 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_  
primary, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  
2025-11-30 01:45:24,842 INFO sqlalchemy.engine.Engine [cached since 0.0203s ago] ('f565305d-9a01-4019-93ff-9c2c76dd1910', 'a3141812-e  
33c-4641-b571-5f272673c22f', '789 Pine Rd', 'Chicago', None, None, 'USA', 1, '2025-11-30 01:45:24.841931', '2025-11-30 01:45:24.84193  
6')  
2025-11-30 01:45:24,843 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u  
ser_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at
```

Скриншот 6, 7.

- 1) выводим alembic current - проверяем, что миграции применены
- 2) проверяем, что таблицы созданы

```
python -c "
```

```
from sqlalchemy import create_engine, inspect
engine = create_engine('sqlite:///lab2.db')
inspector = inspect(engine)
print('Таблицы:', inspector.get_table_names())
)
```

- 3) наполняем БД данными: python seed_data.py

```
2025-11-30 01:45:24,843 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u
sers_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at
FROM users
WHERE users.id = ?
2025-11-30 01:45:24,844 INFO sqlalchemy.engine.Engine [cached since 0.03551s ago] ('90a4654a-578c-40f0-8cca-52c0e254d835',)
2025-11-30 01:45:24,845 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_
primary, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
2025-11-30 01:45:24,846 INFO sqlalchemy.engine.Engine [cached since 0.02391s ago] ('50d8b95f-1c74-438d-9832-b71d9149da05', '90a4654a-
578c-40f0-8cca-52c0e254d835', '321 Elm St', 'Miami', None, None, 'USA', 1, '2025-11-30 01:45:24.845623', '2025-11-30 01:45:24.845627')
2025-11-30 01:45:24,847 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username, users.email AS u
sers_email, users.created_at AS users_created_at, users.updated_at AS users_updated_at
FROM users
WHERE users.id = ?
2025-11-30 01:45:24,847 INFO sqlalchemy.engine.Engine [cached since 0.0391s ago] ('aa787f1d-a530-433e-b8f9-c445c48a6ba0',)
2025-11-30 01:45:24,848 INFO sqlalchemy.engine.Engine INSERT INTO addresses (id, user_id, street, city, state, zip_code, country, is_
primary, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
2025-11-30 01:45:24,849 INFO sqlalchemy.engine.Engine [cached since 0.02702s ago] ('c8494778-3421-4cc9-ae54-7ad6ba505d6e', 'aa787f1d-
a530-433e-b8f9-c445c48a6ba0', '654 Maple Dr', 'Seattle', None, None, 'USA', 1, '2025-11-30 01:45:24.848749', '2025-11-30 01:45:24.848
752')
2025-11-30 01:45:24,849 INFO sqlalchemy.engine.Engine COMMIT
✓ Адреса созданы
🎉 База данных успешно наполнена тестовыми данными!
PS C:\lab2_final> python queries.py
== ПОЛЬЗОВАТЕЛИ С АДРЕСАМИ ==
👤 Пользователь: john_doe (john@example.com)
📍 Адреса:
- 123 Main St, New York, USA
-----
👤 Пользователь: jane_smith (jane@example.com)
📍 Адреса:
- 456 Oak Ave, Los Angeles, USA
-----
👤 Пользователь: bob_wilson (bob@example.com)
📍 Адреса:
- 789 Pine Rd, Chicago, USA
```

Скриншот 8. Тестируем запросы связанных данных: python queries.py

```

a530-433e-b8f9-c445c48a6ba0', '654 Maple Dr', 'Seattle', None, None, 'USA', 1, '2025-11-30 01:45:24.848749', '2025-11-30 01:45:24.848749')
2025-11-30 01:45:24,849 INFO sqlalchemy.engine.Engine COMMIT
✔ Адреса созданы
❗ База данных успешно наполнена тестовыми данными!
PS C:\lab2_final> python queries.py
== ПОЛЬЗОВАТЕЛИ С АДРЕСАМИ ==
👤 Пользователь: john_doe (john@example.com)
📍 Адреса:
- 123 Main St, New York, USA
-----
👤 Пользователь: jane_smith (jane@example.com)
📍 Адреса:
- 456 Oak Ave, Los Angeles, USA
-----
👤 Пользователь: bob_wilson (bob@example.com)
📍 Адреса:
- 789 Pine Rd, Chicago, USA
-----
👤 Пользователь: alice_brown (alice@example.com)
📍 Адреса:
- 321 Elm St, Miami, USA
-----
👤 Пользователь: charlie_davis (charlie@example.com)
📍 Адреса:
- 654 Maple Dr, Seattle, USA
-----
PS C:\lab2_final> python main.py
🔗 ЛАБОРАТОРНАЯ РАБОТА №2: SQLAlchemy и Alembic
=====

Выберите действие:
1 - Наполнить БД тестовыми данными
2 - Вывести связанные данные
3 - Выйти

Ваш выбор:

```

Скриншот 9. Запускаем основную программу: python main.py

Добавление дополнительного строкового поля description:

```

# database_extended.py
from sqlalchemy import Column, String, Boolean, ForeignKey, DateTime, Text,
Float, Integer
from sqlalchemy.orm import DeclarativeBase, relationship
import uuid
from datetime import datetime

class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = 'users'

    id = Column(String(36), primary_key=True, default=lambda:
str(uuid.uuid4()))
    username = Column(String(50), nullable=False, unique=True)
    email = Column(String(100), nullable=False, unique=True)
    description = Column(Text, nullable=True) # Новое поле

```

```
created_at = Column(DateTime, default=datetime.now)
updated_at = Column(DateTime, default=datetime.now,
onupdate=datetime.now)

addresses = relationship("Address", back_populates="user")
orders = relationship("Order", back_populates="user")

class Address(Base):
    __tablename__ = 'addresses'

    id = Column(String(36), primary_key=True, default=lambda:
str(uuid.uuid4()))
    user_id = Column(String(36), ForeignKey('users.id'), nullable=False)
    street = Column(String(200), nullable=False)
    city = Column(String(100), nullable=False)
    state = Column(String(100))
    zip_code = Column(String(20))
    country = Column(String(100), nullable=False)
    is_primary = Column(Boolean, default=False)
    created_at = Column(DateTime, default=datetime.now)
    updated_at = Column(DateTime, default=datetime.now,
onupdate=datetime.now)

    user = relationship("User", back_populates="addresses")
    orders = relationship("Order", back_populates="delivery_address")

class Product(Base):
    __tablename__ = 'products'

    id = Column(String(36), primary_key=True, default=lambda:
str(uuid.uuid4()))
    name = Column(String(100), nullable=False)
    description = Column(Text)
    price = Column(Float, nullable=False)
    stock_quantity = Column(Integer, default=0)
    created_at = Column(DateTime, default=datetime.now)
```

```
updated_at = Column(DateTime, default=datetime.now,
onupdate=datetime.now)

order_items = relationship("OrderItem", back_populates="product")

class Order(Base):
    __tablename__ = 'orders'

    id = Column(String(36), primary_key=True, default=lambda:
str(uuid.uuid4()))
    user_id = Column(String(36), ForeignKey('users.id'), nullable=False)
    delivery_address_id = Column(String(36), ForeignKey('addresses.id'),
nullable=False)
    status = Column(String(50), default='pending')
    total_amount = Column(Float, default=0.0)
    created_at = Column(DateTime, default=datetime.now)
    updated_at = Column(DateTime, default=datetime.now,
onupdate=datetime.now)

    user = relationship("User", back_populates="orders")
    delivery_address = relationship("Address", back_populates="orders")
    order_items = relationship("OrderItem", back_populates="order")

class OrderItem(Base):
    __tablename__ = 'order_items'

    id = Column(String(36), primary_key=True, default=lambda:
str(uuid.uuid4()))
    order_id = Column(String(36), ForeignKey('orders.id'), nullable=False)
    product_id = Column(String(36), ForeignKey('products.id'), nullable=False)
    quantity = Column(Integer, default=1)
    unit_price = Column(Float, nullable=False)
    created_at = Column(DateTime, default=datetime.now)

    order = relationship("Order", back_populates="order_items")
    product = relationship("Product", back_populates="order_items")
```

Ответы на вопросы:

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

- Classical mapping - устаревший подход, явное сопоставление классов с таблицами
- Declarative mapping - современный подход, используя классы-модели, наследуемые от DeclarativeBase
- Imperative mapping - гибридный подход, позволяет явно определять таблицы и сопоставлять их с классами

2. Как Alembic отслеживает текущую версию базы данных?

Alembic создает таблицу `alembic_version` в БД, которая хранит хэш последней примененной миграции.

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

- One-to-Many: User ↔ Address, User ↔ Order, Address ↔ Order
- One-to-One: через ForeignKey с уникальными ограничениями
- Many-to-Many: через промежуточную таблицу OrderItem (Order ↔ Product)

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

Миграция - это процесс изменения схемы БД. Важна для:

- Контроля версий схемы БД
- Безопасного развертывания изменений
- Отслеживания истории изменений
- Совместной работы команды

Как обрабатываются отношения многие-ко-многим в SQLAlchemy?

Через промежуточную таблицу (association table) и relationship с параметром `secondary`.

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

1. Определить текущее состояние `alembic current`
2. Просмотреть историю `alembic history`
3. При необходимости откатиться `alembic downgrade`
4. Разрешить конфликт вручную
5. Применить миграции заново

Вывод

В ходе лабораторной работы были освоены принципы работы с ORM SQLAlchemy и системой миграций Alembic. Были созданы модели данных, реализованы различные типы связей между таблицами, изучены механизмы миграции БД и наполнения данными. Полученные навыки позволяют эффективно работать с реляционными базами данных в Python-приложениях.