

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИМЕНИ ПЕРВОГО ПРЕЗИДЕНТА РОССИИ Б.Н. ЕЛЬЦИНА
(УрФУ имени первого Президента России Б.Н. Ельцина)
Институт радиоэлектроники и информационных технологий — РТФ
Школа профессионального и академического образования

РАБОТА С SQLALCHEMY И ALEMBIC

Отчет по лабораторной работе №2

по дисциплине «Разработка приложений»

	Дата	Подпись	
Преподаватель:	_____	_____	<u>Стаин Д.А.</u>
Студенты:	_____	_____	<u>Пантелеев Е.А.</u>
Группа: РИМ-150950			

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.

ХОД РАБОТЫ

1. Инициализация БД

В начале работы было создано виртуальное окружение python, в которое были установлена библиотека и последующие библиотеки.

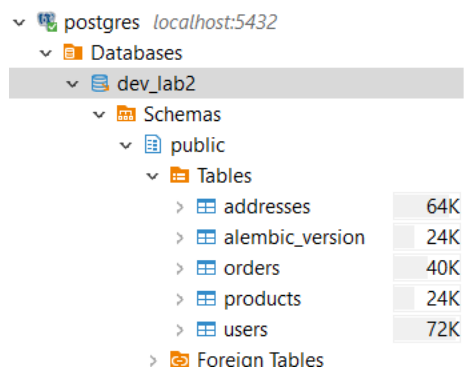
```
(.env) C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2>pip install uv
Collecting uv
  Downloading uv-0.9.3-py3-none-win_amd64.whl.metadata (12 kB)
  Downloading uv-0.9.3-py3-none-win_amd64.whl (21.3 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 5.2 MB/s 0:00:04
Installing collected packages: uv
Successfully installed uv-0.9.3

(.env) C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2>pip list
Package Version
-----
pip      25.2
uv       0.9.3
```

Затем были созданы модели для ORM с пользователями и адресами. Код объявления классов будет приведен в следующих пунктах уже с другими дополнительными классами. В качестве СУБД использовалась локальная PostgreSQL. Была совершена миграция, таблицы создались в БД.

```
(.env) C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2>alembic revision --autogenerate -m "Create users and addresses tables in PostgreSQL"
INFO [alembic.runtime.migration] context impl PostgresqlImpl.
INFO [alembic.autogenerate.compare] Detected added table 'users'
INFO [alembic.autogenerate.compare] Detected added index 'ix_users_email' on 'email',
INFO [alembic.autogenerate.compare] Detected added index 'ix_users_id' on 'id',
INFO [alembic.autogenerate.compare] Detected added index 'ix_users_username' on 'username',
INFO [alembic.autogenerate.compare] Detected added table 'addresses'
INFO [alembic.autogenerate.compare] Detected added index 'ix_addresses_id' on 'id',
INFO [alembic.autogenerate.compare] Detected added index 'ix_addresses_user_id' on 'user_id',
Generating C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2\migrations\versions\f733bd21dda8_create_users_and_addresses_tables_in_py ... done

(.env) C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2>alembic upgrade head
INFO [alembic.runtime.migration] context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade -> f733bd21dda8, Create users and addresses tables in PostgreSQL
```



The screenshot shows the pgAdmin interface for a PostgreSQL database named 'postgres' on localhost:5432. Under the 'Databases' section, 'dev_lab2' is expanded, showing a 'public' schema. Within the 'public' schema, there is a 'Tables' section containing six tables: 'addresses' (64K), 'alembic_version' (24K), 'orders' (40K), 'products' (24K), 'users' (72K), and 'Foreign Tables'.

Database	Schema	Table	Size
postgres	dev_lab2	addresses	64K
postgres	dev_lab2	alembic_version	24K
postgres	dev_lab2	orders	40K
postgres	dev_lab2	products	24K
postgres	dev_lab2	users	72K
postgres	dev_lab2	Foreign Tables	

```

migrations > versions > f733bd21dda8_create_users_and_addresses_tables_in_py > ...
1  """Create users and addresses tables in PostgreSQL
2
3  Revision ID: f733bd21dda8
4  Revises:
5  Create Date: 2025-10-16 14:53:04.264588
6
7  """
8  from typing import Sequence, Union
9
10 from alembic import op
11 import sqlalchemy as sa
12
13 # revision identifiers, used by Alembic.
14 revision: str = 'f733bd21dda8'
15 down_revision: Union[str, Sequence[str], None] = None
16 branch_labels: Union[str, Sequence[str], None] = None
17 depends_on: Union[str, Sequence[str], None] = None
18
19
20
21 GigaCode: explain | explain step by step | doc | test
22 def upgrade() -> None:
23     """Upgrade schema."""
24     # ### commands auto generated by Alembic - please adjust! ###
25     op.create_table('users',
26         sa.Column('id', sa.UUID(), nullable=False),
27         sa.Column('username', sa.String(length=50), nullable=False),
28         sa.Column('email', sa.String(length=100), nullable=False),
29         sa.Column('created_at', sa.DateTime(), nullable=False),
30         sa.Column('updated_at', sa.DateTime(), nullable=False),
31         sa.PrimaryKeyConstraint('id')
32     )
33     op.create_index(op.f('ix_users_email'), 'users', ['email'], unique=True)
34     op.create_index(op.f('ix_users_id'), 'users', ['id'], unique=False)
35     op.create_index(op.f('ix_users_username'), 'users', ['username'], unique=True)
36     op.create_table('addresses',
37         sa.Column('id', sa.UUID(), nullable=False),
38         sa.Column('user_id', sa.UUID(), nullable=False),
39         sa.Column('street', sa.String(length=200), nullable=False),
40         sa.Column('city', sa.String(length=100), nullable=False),
41         sa.Column('state', sa.String(length=100), nullable=False),
42         sa.Column('zip_code', sa.String(length=20), nullable=False),
43         sa.Column('country', sa.String(length=100), nullable=False),
44         sa.Column('is_primary', sa.Boolean(), nullable=False),
45         sa.Column('created_at', sa.DateTime(), nullable=False),
46         sa.Column('updated_at', sa.DateTime(), nullable=False),
47         sa.ForeignKeyConstraint(['user_id'], ['users.id'], ondelete='CASCADE'),
48         sa.PrimaryKeyConstraint('id')
49     )
50     op.create_index(op.f('ix_addresses_id'), 'addresses', ['id'], unique=False)
51     op.create_index(op.f('ix_addresses_user_id'), 'addresses', ['user_id'], unique=False)
52     # ### end Alembic commands ###

```

После этого таблицы были заполнены данными.

```

load_data.py > load_data
1  from sqlalchemy import create_engine
2  from sqlalchemy.orm import sessionmaker
3  from models import User, Address
4  from dotenv import load_dotenv
5  import os
6
7  load_dotenv()
8  DB_URL = os.getenv('DB_URL')
9  engine = create_engine(DB_URL)
10
11 # Создаем фабрику сессий
12 session_factory = sessionmaker(bind=engine)
13
GigaCode: explain | explain step by step | doc | test
14 def load_data():
15     with session_factory() as session:
16
17         user1 = User(
18             username="ivanov",
19             email="ivanov@example.com"
20         )
21         address1_1 = Address(
22             street="ул. Ленина, 15",
23             city="Москва",
24             state="Московская область",
25             zip_code="101000",
26             country="Russia",
27             is_primary=True,
28             user=user1
29         )
30         address1_2 = Address(
31             street="пр. Победы, 42",
32             city="Москва",
33             state="Московская область",
34             zip_code="101001",
35             country="Russia",
36             is_primary=False,
37             user=user1
38         )
39
40         user2 = User(
41             username="petrov",
42             email="petrov@example.com"
43         )
44         address2_1 = Address(
45             street="Невский пр., 25",
46             city="Санкт-Петербург",
47             state="Ленинградская область",
48             zip_code="190000",
49             country="Russia",
50             is_primary=True,
51             user=user2
52         )

```

```
53
54     user3 = User(
55         username="sidorova",
56         email="sidorova@example.com"
57     )
58     address3_1 = Address(
59         street="ул. Баумана, 8",
60         city="Казань",
61         state="Татарстан",
62         zip_code="420000",
63         country="Russia",
64         is_primary=True,
65         user=user3
66     )
67     address3_2 = Address(
68         street="ул. Кремлевская, 35",
69         city="Казань",
70         state="Татарстан",
71         zip_code="420001",
72         country="Russia",
73         is_primary=False,
74         user=user3
75     )
76
77     user4 = User(
78         username="kozlov",
79         email="kozlov@example.com"
80     )
81     address4_1 = Address(
82         street="ул. Советская, 77",
83         city="Новосибирск",
84         state="Новосибирская область",
85         zip_code="630000",
86         country="Russia",
87         is_primary=True,
88         user=user4
89     )
90
91     user5 = User(
92         username="novikova",
93         email="novikova@example.com"
94     )
95     address5_1 = Address(
96         street="ул. Красная, 12",
97         city="Краснодар",
98         state="Краснодарский край",
99         zip_code="350000",
100         country="Russia",
101         is_primary=True,
102         user=user5
103     )
```

```

104     address5_2 = Address(
105         street="ул. Мира, 33",
106         city="Сочи",
107         state="Краснодарский край",
108         zip_code="354000",
109         country="Russia",
110         is_primary=False,
111         user=user5
112     )
113     address5_3 = Address(
114         street="ул. Курортная, 5",
115         city="Анапа",
116         state="Краснодарский край",
117         zip_code="353440",
118         country="Russia",
119         is_primary=False,
120         user=user5
121     )
122
123     users = [user1, user2, user3, user4, user5]
124     addresses = [
125         address1_1, address1_2, address2_1, address3_1, address3_2,
126         address4_1, address5_1, address5_2, address5_3
127     ]
128
129     session.add_all(users + addresses)
130     session.commit()
131     print("Данные успешно добавлены!")
132
133 if __name__ == "__main__":
134     load_data()

```

2. Запрос связанных данных

Для запроса данных был написан простенький запрос, который извлекает юзеров с их адресами.

```

queries.py > ...
1  from sqlalchemy import select
2  from sqlalchemy.orm import selectinload, Session
3  from load_data import engine
4  from models import User, Address
5
6  def get_users_addresses():
7      """Получить всех пользователей с адресами"""
8      stmt=select(User).options(selectinload(User.addresses))
9
10     with Session(engine) as session:
11         users = session.execute(stmt).scalars().all()
12
13         for user in users:
14             print(f"👤 {user.username} ({user.email})")
15             for address in user.addresses:
16                 print(f"    {address.city}, {address.street}")
17             print()
18
19     return users
20
21 if __name__ == "__main__":
22     get_users_addresses()
23

```

Ну и собственно результат запроса представлен ниже.

```
(.venv) C:\Users\Euvgene\Desktop\Уник\Разработка приложений (Кузьмин Д)\lab2>python queries.py
👤 ivanov (ivanov@example.com)
Москва, ул. Ленина, 15
Москва, пр. Победы, 42

👤 petrov (petrov@example.com)
Санкт-Петербург, Невский пр., 25

👤 sidorova (sidorova@example.com)
Казань, ул. Баумана, 8
Казань, ул. Кремлевская, 35

👤 kozlov (kozlov@example.com)
Новосибирск, ул. Советская, 77

👤 novikova (novikova@example.com)
Краснодар, ул. Красная, 12
Сочи, ул. Мира, 33
Анапа, ул. Курортная, 5
```

3. Последующие работы с БД и миграции

Затем задание было следующим:

«В ОРМ добавляем пользователю дополнительное строковое поле `description`. Добавляем дополнительную таблицу для продукции и заказов. Заказ должен в себе содержать информацию о пользователе, адресе доставки и продукции. Производим миграцию данных и добавляем 5 продуктов и 5 заказов в БД.»

Для начала был изменен файл `models.py`, привожу скриншоты сразу всех моделей.

```

models.py > User
1  from sqlalchemy.orm import DeclarativeBase, relationship, Mapped, mapped_column
2  from sqlalchemy import ForeignKey, String, Boolean, DateTime, Numeric, Text
3  from typing import List, Optional
4  from decimal import Decimal
5  from sqlalchemy.dialects.postgresql import UUID
6  from datetime import datetime
7  import uuid
8
9  GigaCode: explain | explain step by step | doc | test
10 class Base(DeclarativeBase):
11     pass
12
13 GigaCode: explain | explain step by step | doc | test
14 class User(Base):
15     __tablename__ = "users"
16
17     id: Mapped[uuid.UUID] = mapped_column(
18         UUID(as_uuid=True),
19         primary_key=True,
20         default=uuid.uuid4,
21         index=True)
22
23     username: Mapped[str] = mapped_column(String(50), nullable=False, unique=True, index=True)
24     email: Mapped[str] = mapped_column(String(100), nullable=False, unique=True, index=True)
25     created_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
26     updated_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
27     description: Mapped[str] = mapped_column(String(300), nullable=True)
28
29     addresses: Mapped[List["Address"]] = relationship("Address", back_populates="user")
30
31 GigaCode: explain | explain step by step | doc | test
32 class Address(Base):
33     __tablename__ = 'addresses'
34
35     id: Mapped[uuid.UUID] = mapped_column(
36         UUID(as_uuid=True),
37         primary_key=True,
38         default=uuid.uuid4,
39         index=True)
40
41     user_id: Mapped[uuid.UUID] = mapped_column(
42         UUID(as_uuid=True),
43         ForeignKey('users.id', ondelete='CASCADE'),
44         nullable=False,
45         index=True)
46
47     street: Mapped[str] = mapped_column(String(200), nullable=False)
48     city: Mapped[str] = mapped_column(String(100), nullable=False)
49     state: Mapped[str] = mapped_column(String(100))
50     zip_code: Mapped[str] = mapped_column(String(20))
51     country: Mapped[str] = mapped_column(String(100), nullable=False)
52     is_primary: Mapped[bool] = mapped_column(Boolean, default=False)
53     created_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
54     updated_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
55     user: Mapped["User"] = relationship("User", back_populates="addresses")

```



```

models.py > User
GigaCode: explain | explain step by step | doc | test
55 class Product(Base):
56     __tablename__ = "products"
57
58     id: Mapped[uuid.UUID] = mapped_column(
59         UUID(as_uuid=True),
60         primary_key=True,
61         default=uuid.uuid4,
62         index=True)
63
64     name: Mapped[str] = mapped_column(String(100), nullable=False)
65     description: Mapped[Optional[str]] = mapped_column(Text, nullable=True)
66     price: Mapped[Decimal] = mapped_column(Numeric(10, 2), nullable=False)
67     created_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
68     updated_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
69
70 class Order(Base):
71     __tablename__ = "orders"
72
73     id: Mapped[uuid.UUID] = mapped_column(
74         UUID(as_uuid=True),
75         primary_key=True,
76         default=uuid.uuid4,
77         index=True)
78
79     user_id: Mapped[uuid.UUID] = mapped_column(
80         UUID(as_uuid=True),
81         ForeignKey('users.id', ondelete='CASCADE'),
82         nullable=False,
83         index=True)
84
85     address_id: Mapped[uuid.UUID] = mapped_column(
86         UUID(as_uuid=True),
87         ForeignKey('addresses.id', ondelete='CASCADE'),
88         nullable=False,
89         index=True)
90
91     product_id: Mapped[uuid.UUID] = mapped_column(
92         UUID(as_uuid=True),
93         ForeignKey('products.id', ondelete='CASCADE'),
94         nullable=False,
95         index=True)
96
97     quantity: Mapped[int] = mapped_column(nullable=False, default=1)
98     status: Mapped[str] = mapped_column(String(50), default="pending")
99     total_amount: Mapped[Decimal] = mapped_column(Numeric(10, 2), nullable=False)
100     created_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
101     updated_at: Mapped[datetime] = mapped_column(DateTime, default=datetime.now)
102
103     user: Mapped['User'] = relationship('User')
104     address: Mapped['Address'] = relationship('Address')
105     product: Mapped['Product'] = relationship('Product')

```

После этого была выполнена миграция в alembic, прилагаю скриншот файла миграции. Как мы видим здесь только добавление новых моделей и столбца description.

```

migrations > versions > 432ceed6091b_add_description_and_products_orders.py > ...
1  """add_description_and_products_orders
2
3  Revision ID: 432ceed6091b
4  Revises: f733bd21dda8
5  Create Date: 2025-10-16 17:06:35.093414
6
7  """
8  from typing import Sequence, Union
9
10 from alembic import op
11 import sqlalchemy as sa
12
13
14 # revision identifiers, used by Alembic.
15 revision: str = '432ceed6091b'
16 down_revision: Union[str, Sequence[str], None] = 'f733bd21dda8'
17 branch_labels: Union[str, Sequence[str], None] = None
18 depends_on: Union[str, Sequence[str], None] = None
19
20
21 def upgrade() -> None:
22     """Upgrade schema."""
23     # ### commands auto generated by Alembic - please adjust! ###
24     op.create_table('products',
25         sa.Column('id', sa.UUID(), nullable=False),
26         sa.Column('name', sa.String(length=100), nullable=False),
27         sa.Column('description', sa.Text(), nullable=True),
28         sa.Column('price', sa.Numeric(precision=10, scale=2), nullable=False),
29         sa.Column('created_at', sa.DateTime(), nullable=False),
30         sa.Column('updated_at', sa.DateTime(), nullable=False),
31         sa.PrimaryKeyConstraint('id')
32     )
33     op.create_index(op.f('ix_products_id'), 'products', ['id'], unique=False)
34     op.create_table('orders',
35         sa.Column('id', sa.UUID(), nullable=False),
36         sa.Column('user_id', sa.UUID(), nullable=False),
37         sa.Column('address_id', sa.UUID(), nullable=False),
38         sa.Column('product_id', sa.UUID(), nullable=False),
39         sa.Column('quantity', sa.Integer(), nullable=False),
40         sa.Column('status', sa.String(length=50), nullable=False),
41         sa.Column('total_amount', sa.Numeric(precision=10, scale=2), nullable=False),
42         sa.Column('created_at', sa.DateTime(), nullable=False),
43         sa.Column('updated_at', sa.DateTime(), nullable=False),
44         sa.ForeignKeyConstraint(['address_id'], ['addresses.id'], ondelete='CASCADE'),
45         sa.ForeignKeyConstraint(['product_id'], ['products.id'], ondelete='CASCADE'),
46         sa.ForeignKeyConstraint(['user_id'], ['users.id'], ondelete='CASCADE'),
47         sa.PrimaryKeyConstraint('id')
48     )
49     op.create_index(op.f('ix_orders_address_id'), 'orders', ['address_id'], unique=False)
50     op.create_index(op.f('ix_orders_id'), 'orders', ['id'], unique=False)
51     op.create_index(op.f('ix_orders_product_id'), 'orders', ['product_id'], unique=False)
52     op.create_index(op.f('ix_orders_user_id'), 'orders', ['user_id'], unique=False)
53     op.add_column('users', sa.Column('description', sa.String(length=300), nullable=True))
54     # ### end Alembic commands ###
55

```

Заполнение таблицы было выполнено с помощью функций.

```

update_data.py > ...
1  # update_data.py
2  from sqlalchemy import select
3  from sqlalchemy.orm import Session, selectinload
4  from load_data import session_factory
5  from models import User, Product, Order, Address
6  from decimal import Decimal
7
8  GigaCode: explain | explain step by step | doc | test
9  def update_users_with_descriptions():
10     """Добавляет описания существующим пользователям"""
11     with session_factory() as session:
12         users = session.execute(select(User)).scalars().all()
13
14         descriptions = [
15             "Любитель путешествий и фотографии",
16             "Программист и геймер",
17             "Дизайнер интерьеров из Казани",
18             "Студент университета в Новосибирске",
19             "Предприниматель из Краснодарского края"
20         ]
21
22         for user, description in zip(users, descriptions):
23             user.description = description
24
25         session.commit()
26         print("Описания добавлены!")

```

```

update_data.py > ...
GigaCode: explain | explain step by step | doc | test
27 def add_products_and_orders():
28     """Добавляет продукты и заказы"""
29     with session_factory() as session:
30
31         products = [
32             Product(name="Ноутбук Gaming Pro", description="Игровой ноутбук с RTX 4060", price=Decimal("150000.00")),
33             Product(name="Смартфон Galaxy S24", description="Флагманский смартфон", price=Decimal("90000.00")),
34             Product(name="Наушники Wireless", description="Беспроводные наушники с шумоподавлением", price=Decimal("15000.00")),
35             Product(name="Умные часы Pro", description="Смарт-часы с функцией ECG", price=Decimal("50000.00")),
36             Product(name="Планшет для рисования", description="Графический планшет с пером", price=Decimal("30000.00"))
37         ]
38
39         session.add_all(products)
40         session.flush()
41         print("Продукты созданы!")
42
43         users = session.execute(
44             select(User).options(selectinload(User.addresses))
45         ).scalars().all()
46
47         orders = []
48         for i in range(5):
49             user = users[i]
50
51             primary_address = next((addr for addr in user.addresses if addr.is_primary), None)
52
53             if not primary_address and user.addresses:
54                 primary_address = user.addresses[0]
55                 print(f"Пользователя {user.username} нет основного адреса, взят первый")
56
57             if not primary_address:
58                 print(f"Пользователя {user.username} нет адресов, пропускаем заказ")
59                 continue
60
61             order = Order(
62                 user_id=user.id,
63                 address_id=primary_address.id,
64                 product_id=products[i].id,
65                 quantity=1,
66                 status="completed",
67                 total_amount=products[i].price
68             )
69             orders.append(order)
70
71         session.add_all(products + orders)
72         session.commit()
73         print("Продукты и заказы добавлены!")
74
75 if __name__ == "__main__":
76     update_users_with_descriptions()
77     add_products_and_orders()

```

Пример таблицы заказов.

jack

Auto

postgres

public@dev_lab2

users

addresses

orders

products

Properties

Data

ER Diagram

orders

Enter a SQL expression to filter results (use Ctrl+Space)

	address_id	product_id	quantity	status	total_amount	created_at	updated_at
1	06-5a62eb6b30aa	2c173974-dba8-41c2-a2a9-9efb0c1947c6	b7290164-f3ea-4991-8eca-a1018d290a6a	1 completed	150,000	2025-10-16 17:47:08.051	2025-10-16 17:47:08.051
2	15a-1f26a3826436	b7e0f469-470e-424c-b342-671fb62969aa	2cd39fc3-870f-4a26-913f-6863047ac89f	1 completed	90,000	2025-10-16 17:47:08.051	2025-10-16 17:47:08.051
3	50-5f8990b1dd9e	986d6128-2498-4544-b05f-0abd4a1fc1ca	363d6f87-c88c-44d3-8dae-9a8eade4fd77	1 completed	15,000	2025-10-16 17:47:08.051	2025-10-16 17:47:08.051
4	a1-a79d9ad5f1e1	9b2ad658-b45a-4617-b628-4f792a95a6b0	46aa23f4-3950-4e37-96eb-e91c48e490fb	1 completed	50,000	2025-10-16 17:47:08.051	2025-10-16 17:47:08.051
5	a-007e9768ce66	5c8f1d29-7e24-4d0b-80f2-dbfaf54253f7	eaca1f48-905f-4c5a-9e1a-783c449065d4	1 completed	30,000	2025-10-16 17:47:08.051	2025-10-16 17:47:08.051

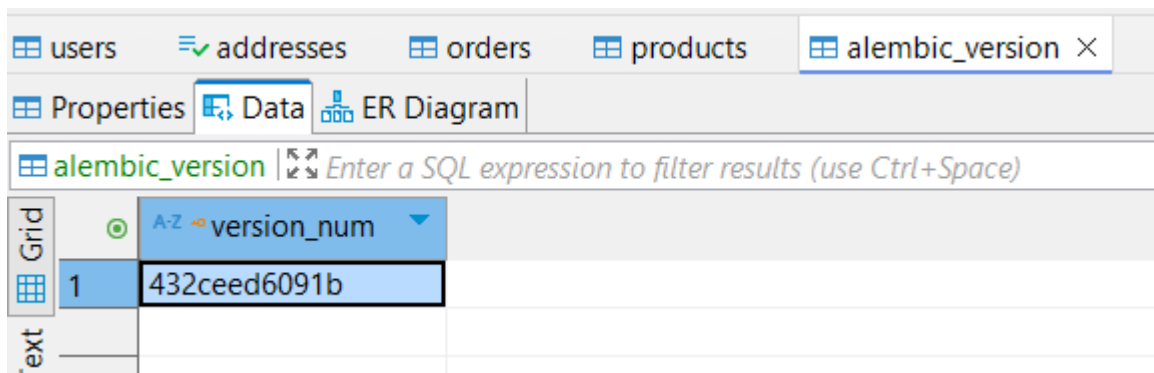
4. Ответы на вопросы

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

Есть следующие подходы: императивный и декларативный. В императивном сначала определяется схема таблицы, а потом она связывается с python-классом. Следует использовать, если нужно больше контроля или для легаси кода. В декларативном класс является и моделью данных и таблицей. Является более используемым и более лаконичным точки зрения кода.

2. Как Alembic отслеживает текущую версию базы данных?

По таблице в БД alembic_version.



	version_num
1	432ceed6091b

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

Один ко многим – юзер имеет несколько адресов

Многие к одному – заказ имеет связи до юзера, адреса и товара

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

Миграция – это изменение структуры БД в ходе развития БД или приложения. Она важна, потому что нужно отслеживать версии и иметь возможность их откатить, особенно, когда есть большая команда разработчиков и большой сервис.

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

Через ассоциативную таблицу

```

main.py > User
1  user_product_association = Table(
2      'user_product_association', Base.metadata,
3      Column('user_id', ForeignKey('users.id'), primary_key=True),
4      Column('product_id', ForeignKey('products.id'), primary_key=True),
5      Column('purchased_at', DateTime, default=datetime.now)
6  )
7
GigaCode: explain | explain step by step | doc | test
8  class User(Base):
9      products: Mapped[List["Product"]] = relationship(
10         "Product",
11         secondary=user_product_association,
12         back_populates="users"
13     )

```

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

- Определить текущее состояние (alembic_current)
- Посмотреть историю, чтобы понять где ветки разошлись
- Сделать слияние
- Обновить БД

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были освоены принципы работы с библиотеками SQLAlchemy и Alembic для создания и управления реляционными базами данных на Python и изучены механизмы миграции базы данных.