

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
з Лабораторної роботи №3
із дисципліни «Бази даних»
на тему
«Міграція Бази Даних»

Виконав студент групи КМ-22:

Ляпко Кирило

Керівник:

Жук І. С.

Зміст

Вступ	3
1.1 Мета роботи	3
1.2 Постановка задачі	3
Основна частина	4
2.1 Створення ORM-моделі	4
2.2 Міграції	6
2.3 Створення інтерфейсу	9
2.4 Міграція з обнієї бази данної до іншої	12

Вступ

1.1 Мета роботи.

Метою даної роботи є набуття практичних навичок розробки, рефакторингу та міграції бази даних погодних умов у стилі Layered Architecture, з використанням ORM (Object-Relational Mapping), інструментів міграції схем БД, а також переходу між різними типами баз даних.

1.2 Постановка задачі.

Завдання: написати програму в стилі Layered Architecture, в якій би була описана структура БД за допомогою ORM моделі.

1. Відібрати колонки, які будуть описані в ORM. Серед колонок мають бути в кількості хоча б одна текстова (country), з цілим числом (wind_degree), з дробним числом (wind_kph), з enumation (wind_direction), з датою (last_updated) та з часом (sunrise).

2. Також взяти в опрацювання всі колонки стосовно однієї категорії для подальшого оцінювання в залежності від кратності варіанта вашому номеру в списку. Оскільки, мій варіант - 15, а він кратний одиниці і трьом, то я обираю колонку вітру та осадків.

3. Щоб погодні дані не були всі в одній купі, треба впорядкувати БД - зробити рефакторинг. Для цього дані стосовно вашої категорії необхідно винести в окрему таблицю. Це робиться інструментом для міграції, наприклад, Flyway, Liquibase, Alembic

4. Створити нову колонку у новоствореній таблиці типу булеан, яка б відображала відповідь на питання “Чи варто виходити на вулицю”. Для заповнення колонки придумайте формулу, яка б на виході базуючись на ваших даних видавала відповіді “так” чи “ні”. Наприклад, вітер більше 10 м/с, краще не виходити.

5. Дайте можливість користувачу задавши країну та дату, можливо ще додаткові параметри, побачити всю інформацію стосовно погоди, яка у вас є. Інтерфейс не обов'язковий, достатньо консолі.

6. Мігрувати з однієї БД - Postgres на іншу - MySQL, або навпаки. Таким чином, всі скрипти, які ви написали мають накотитись на нову базу без проблем, та ви матимете 2 однакові бази.

Основна частина

2.1 Створення ORM-моделі.

Для початку нам необхідно налаштувати та встановити все для ORM-моделі:

```
from sqlalchemy import (
    create_engine, Column, Integer, Float, String, Enum,
    Date, Time, Boolean, ForeignKey, text
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
import enum
```

```
Base = declarative_base()
```

Далі перейдемо вже до виконання лабораторної роботи, а саме оголошення ORM-моделей та розбиття її одразу на три частини:

```
class WindDirectionEnum(enum.Enum):
```

```
    N = 'N'
    NNE = 'NNE'
    NE = 'NE'
    ENE = 'ENE'
    E = 'E'
    ESE = 'ESE'
    SE = 'SE'
    SSE = 'SSE'
    S = 'S'
    SSW = 'SSW'
    SW = 'SW'
    WSW = 'WSW'
    W = 'W'
    WNW = 'WNW'
    NW = 'NW'
    NNW = 'NNW'
```

```
class WindData(Base):
```

```
    __tablename__ = 'wind_data'
```

```
    id = Column(Integer, primary_key = True)
```

```
    weather_id = Column(Integer, ForeignKey('weather.id'), nullable=False)
```

```
    degree = Column(Integer, nullable = False)
```

```
    kph = Column(Float, nullable = False)
```

```

direction = Column(Enum(WindDirectionEnum), nullable = False)
go_out    = Column(Boolean, nullable = False)

weather   = relationship("Weather", back_populates = "wind")
class PrecipitationData(Base):
    __tablename__ = "precipitation_data"

    id = Column(Integer, primary_key = True)
    weather_id = Column(Integer, ForeignKey('weather.id'), nullable=False)
    pressure_mb = Column(Float, nullable = False)
    pressure_in = Column(Float, nullable = False)
    precip_mm = Column(Float, nullable = False)
    precip_in = Column(Float, nullable = False)
    humidity = Column(Integer, nullable = False)
    cloud = Column(Integer, nullable = False)

    weather = relationship("Weather", back_populates = "precipitations")

class Weather(Base):
    __tablename__ = 'weather'

    id          = Column(Integer, primary_key = True)
    country     = Column(String, nullable = False)
    location_name = Column(String, nullable = False)
    last_updated = Column(Date, nullable = False)
    sunrise     = Column(Time, nullable = False)

    wind        = relationship("WindData", back_populates = "weather",
uselist = False)
    precipitations = relationship("PrecipitationData",
back_populates="weather", uselist = False)

```

Оголосивши ORM-моделі ми отримали три таблиці Weather, PrecipitationData та WindData. Weather зберігає основні дані про країну, локацію, дату останнього оновлення та годину сходу сонця. WindData прив'язана Weather через зовнішній ключ і містить інформацію про вітер: градус, швидкість у км/год, напрямок і логічне поле, чи рекомендовано виходити надвір. PrecipitationData зберігає дані, пов'язані з атмосферними опадами та станом повітря: атмосферний тиск у мілібарах і дюймах, кількість опадів у міліметрах і дюймах, відсоткову вологість повітря та ступінь хмарності. Ця таблиця також прив'язана до Weather через зовнішній ключ weather_id, що забезпечує відповідність даних одній погодній події в конкретному місці та на конкретну дату.

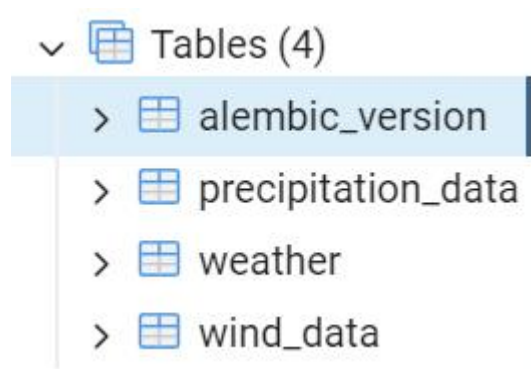
Також між моделями створено двосторонній зв'язок типу «один до одного» (one-to-one), що спрощує навігацію між основними погодними даними та інформацією про вітер і опади. Цей зв'язок реалізовано за допомогою зовнішнього ключа (ForeignKey) у моделях WindData та PrecipitationData, які посилаються на таблицю Weather.

2.2 Міграції.

Далі оскільки ми використовуємо мову програмування Python, то щоб дані винести в окремі таблиці нам необхідно налаштувати правильним чином Alembic, зробимо це:

- Спочатку нам необхідно його встановити, це можна зробити за допомогою наступної команди: `pip install alembic`;
- Далі нам необхідн створити файли `alembic.ini` та `env.py` в нашій папці з кодом: `alembic init alembic`;
- Тепер нам необхідно додати url до нашої бази даних у файлі `alembic.ini` та замінити на наш клас Base у файлі `env.py`;
- І на останок запускаємо два скрипти, перший: `alembic revision --autogenerate -m "create weather schema"`, другий: `alembic upgrade head`.

В результаті цих дій міграція пройшла успішно:



Після цього нам необхідно створити ще одну таблицю типу булеан, яка б відображала відповідь на питання “Чи варто виходити на вулицю”. Для заповнення колонки необхідно придумати формулу, яка б на виході базуючись на ваших даних видавала відповіді “так” чи “ні”.

У нас вона вже створена: `go_out = Column(Boolean, nullable = False)`

Далі перенесемо всі дані заданого датасету до нашої бази даних:

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from models import Base, Weather, WindData, PrecipitationData
DATABASE_URL = "postgresql://kyrylo:157329@localhost:5432/lab3"
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind = engine)
session = Session()
Base.metadata.create_all(engine)
file_path =
r"C:\Users\user\Documents\database\database_labwork_3\code\GlobalWeather
Repository.csv"
df = pd.read_csv(file_path)

for _, row in df.iterrows():
    weather = Weather(
        country = row["country"],
        location_name = row["location_name"],
        last_updated = row["last_updated"],
        sunrise = row["sunrise"]
    )
    session.add(weather)
    session.commit()

    wind = WindData(
        id = weather.id,
        degree = row["wind_degree"],
        kph = row["wind_kph"],
        direction = row["wind_direction"],
        go_out = row["wind_kph"] <= 10
    )
    session.add(wind)

    precipitation = PrecipitationData(
        weather_id = weather.id,
        pressure_mb = row["pressure_mb"],
        pressure_in = row["pressure_in"],
        precip_mm = row["precip_mm"],
        precip_in = row["precip_in"],
        humidity = row["humidity"],
        cloud = row["cloud"]
    )
```

```
session.add(precipitation)
session.commit()
```

Дані були успішно перенесено:

id [PK] integer	country character varying	location_name character varying	last_updated date	sunrise time without time zone
1	Afghanistan	Kabul	2024-05-16	04:50:00
2	Albania	Tirana	2024-05-16	05:21:00
3	Algeria	Algiers	2024-05-16	05:40:00
4	Andorra	Andorra La Vella	2024-05-16	06:31:00
5	Angola	Luanda	2024-05-16	06:12:00
6	Antigua and Barbuda	Saint John's	2024-05-16	05:36:00
7	Argentina	Buenos Aires	2024-05-16	07:43:00
8	Armenia	Yerevan	2024-05-16	05:45:00
9	Australia	Canberra	2024-05-16	06:52:00

weather_id integer	pressure_mb double precision	pressure_in double precision	precip_mm double precision	precip_in double precision	humidity integer	cloud integer
1	1012	29.89	0	0	24	30
2	1012	29.88	0.1	0	94	75
3	1011	29.85	0	0	29	0
4	1007	29.75	0.3	0.01	61	100
5	1011	29.85	0	0	89	50
6	1013	29.91	0.02	0	84	25
7	1014	29.94	0	0	93	0
8	1017	30.03	0.13	0.01	40	25
9	1027	30.33	0	0	87	0

id [PK] integer	weather_id integer	degree integer	kph double precision	direction winddirectionenum	go_out boolean
1	1	338	13.3	NNW	false
2	2	320	11.2	NW	false
3	3	280	15.1	W	false
4	4	215	11.9	SW	false
5	5	150	13	SSE	false
6	6	90	9	E	true
7	7	10	3.6	N	true
8	8	140	6.8	SE	true
9	9	100	4	E	true
10	10	110	20.2	ESE	false
11	11	20	6.8	NNE	true

2.3 Створення інтерфейсу.

Тепер згідно постановки задачі нам необхідно створити інтерфейс де користувач задавши країну та дату, можливо ще додаткові параметри, може побачити всю інформацію стосовно погоди:

```
from models import Base, Weather, WindData, PrecipitationData
from datetime import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "postgresql://kyrylo:157329@localhost:5432/lab3"

engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

def get_valid_country(session):
    while True:
        country = input("Enter country: ").strip()
        exists = session.query(Weather).filter(Weather.country ==
country).first()
        if exists:
            return country
        print(f"No data found for country '{country}'. Please try again.")

def get_date_input():
    while True:
        date_str = input("Enter date (YYYY-MM-DD): ").strip()
        try:
            return datetime.strptime(date_str, "%Y-%m-%d")
        except ValueError:
            print("Invalid date format. Please enter the date in YYYY-MM-DD format.")

def fetch_weather_data(session, country, date):
    return session.query(Weather).join(WindData).filter(
        Weather.country == country,
        Weather.last_updated == date
    ).all()

def display_weather_info(weather, session):
    print(f"\n Country: {weather.country}")
    print(f" Last Updated: {weather.last_updated}")
    print(f" Sunrise: {weather.sunrise}")
```

```

    wind_data = session.query(WindData).filter(WindData.weather_id ==
weather.id).all()
    for wind in wind_data:
        print(f" Wind Direction: {wind.direction}")
        print(f" Wind Speed (kph): {wind.kph}")
        print(f" Wind Angle: {wind.degree}°")
        print(f" Should Go Outside: {'Yes' if wind.go_out else 'No'}")

    precipitation_data = session.query(PrecipitationData).filter(
        PrecipitationData.weather_id == weather.id
    ).all()
    for p in precipitation_data:
        print(f" Pressure (mb): {p.pressure_mb}")
        print(f" Pressure (inches Hg): {p.pressure_in}")
        print(f" Precipitation (mm): {p.precip_mm}")
        print(f" Precipitation (inches): {p.precip_in}")
        print(f" Humidity: {p.humidity}%")
        print(f" Cloudiness: {p.cloud}%")

def interface():
    session = Session()
    try:
        country = get_valid_country(session)
        date = get_date_input()
        weather_data = fetch_weather_data(session, country, date)

        if not weather_data:
            print(f"No records found for country '{country}' on {date.date()}.")
            return

        for weather in weather_data:
            display_weather_info(weather, session)

    finally:
        session.close()

if __name__ == "__main__":
    interface()

```

Результат:

```
Enter country: Afghanistan
Enter date (YYYY-MM-DD): 2024-05-16

Country: Afghanistan
Last Updated: 2024-05-16
Sunrise: 04:50:00
Wind Direction: WindDirectionEnum.NNW
Wind Speed (kph): 13.3
Wind Angle: 338°
Should Go Outside: No
Pressure (mb): 1012.0
Pressure (inches Hg): 29.89
Precipitation (mm): 0.0
Precipitation (inches): 0.0
Humidity: 24%
Cloudiness: 30%

Country: Afghanistan
Last Updated: 2024-05-16
Sunrise: 04:50:00
Wind Direction: WindDirectionEnum.NW
Wind Speed (kph): 7.2
Wind Angle: 318°
Should Go Outside: Yes
Pressure (mb): 1014.0
Pressure (inches Hg): 29.93
Precipitation (mm): 0.03
Precipitation (inches): 0.0
Humidity: 41%
Cloudiness: 89%
```

2.4 Міграція з однієї бази данної до іншої.

В цій частині роботи, нам необхідно написати код, щоб мігрувати з однієї БД - Postgres на іншу - MySQL, або навпаки. Таким чином, всі скрипти, які ви написали мають накотитись на нову базу без проблем, та ви матимете 2 однакові баз:

Для цього створимо відповідну базу даних використовуючи MySQL і потім додамо новий url до нашої нової бази даних у файлі alembic.ini.

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from models import Base, Weather, WindData, PrecipitationData

# PostgreSQL
POSTGRES_URL = "postgresql://kyrylo:157329@localhost:5432/lab3"
postgres_engine = create_engine(POSTGRES_URL)
PostgresSession = sessionmaker(bind=postgres_engine)
postgres_session = PostgresSession()

# MySQL
MYSQL_URL =
"mysql+pymysql://kyrylo:157329@localhost:3306/lab3"
mysql_engine = create_engine(MYSQL_URL)
MySQLSession = sessionmaker(bind=mysql_engine)
mysql_session = MySQLSession()

# Create tables in both databases
Base.metadata.create_all(postgres_engine)
Base.metadata.create_all(mysql_engine)

def parse_time(t_str):
    from datetime import datetime
    return datetime.strptime(t_str, '%I:%M %p').time()

# Load CSV
file_path =
r"C:\Users\user\Documents\database\database_labwork_3\code\GlobalWeather
Repository.csv"
df = pd.read_csv(file_path)

def insert_data(session):
    for _, r in df.iterrows():
        weather = Weather(
```

```

        country=r["country"],
        location_name=r["location_name"],
        last_updated=r["last_updated"],
        sunrise=parse_time(r["sunrise"])
    )
    session.add(weather)
    session.commit()

    wind = WindData(
        weather_id=weather.id,
        degree=r["wind_degree"],
        kph=r["wind_kph"],
        direction=r["wind_direction"],
        go_out=r["wind_kph"] <= 10
    )
    session.add(wind)




    precipitation = PrecipitationData(
        weather_id=weather.id,
        pressure_mb=float(r["pressure_mb"]),
        pressure_in=float(r["pressure_in"]),
        precip_mm=float(r["precip_mm"]),
        precip_in=float(r["precip_in"]),
        humidity=float(r["humidity"]),
        cloud=int(r["cloud"])
    )
    session.add(precipitation)

    session.commit()
# Insert into MySQL
insert_data(mysql_session)

# Insert into PostgreSQL
insert_data(postgres_session)

```

Результат:

sult Grid   Filter Rows: <input type="text"/> Edit: 					
id	weather_id	degree	kph	direction	go_out
1	1	338	13.3	NNW	0
2	2	320	11.2	NW	0
3	3	280	15.1	W	0
4	4	215	11.9	SW	0
5	5	150	13	SSE	0
6	6	90	9	E	1
7	7	10	3.6	N	1
8	8	140	6.8	SE	1
9	9	100	4	E	1
10	10	110	20.2	ESE	0
11	11	20	6.8	NNE	1
12	12	180	25.9	S	0
13	13	140	22	SE	0
14	14	222	6.8	SW	1
15	15	90	25.9	E	0
16	16	150	19.1	SSE	0
17	17	240	6.1	WSW	1
18	18	99	6.8	E	1
19	19	210	24.1	SSW	0

Висновок

У ході лабораторної роботи було опрацьовано процес управління схемою бази даних із застосуванням інструментів SQLAlchemy та Alembic. Було створено структуру бази даних для зберігання погодних даних, зокрема, таблиці для зберігання інформації про вітер із використанням ENUM-типів для напрямку вітру.

Виконання автоматичних міграцій за допомогою Alembic продемонструвало ефективність у підтримці актуальної схеми бази даних та автоматичному оновленні структури без втрати даних. Під час роботи були виявлені й усунені типові помилки, такі як дублювання типів ENUM, що виникали при повторному запуску міграцій. Це дозволило краще зрозуміти особливості роботи з типами даних у PostgreSQL та способи контролю змін структури бази.

Отримані знання є корисними для побудови масштабованих проєктів, де необхідна гнучкість і контроль версій бази даних. Впровадження міграцій забезпечує безпечне внесення змін у структуру бази, що є критично важливим при роботі з реальними проєктами.