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: | Керівник: |
| Ляпко Кирило | Жук І. С. |

Київ — 2025

**Зміст**

[Вступ 3](#_Toc6231)

[1.1 Мета роботи. 3](#_Toc18485)

[1.2 Постановка задачі. 3](#_Toc11285)

[Основна частина 4](#_Toc10937)

[2.1 Створення ORM-моделі. 4](#_Toc25074)

[2.2 Міграції. 6](#_Toc19721)

[2.3 Створення інтерфейсу. 9](#_Toc6894)

[2.4 Міграція з обнієї бази данної до іншої. 12](#_Toc32623)

**Вступ**

* 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. 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 еnumetaion (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/c, 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.

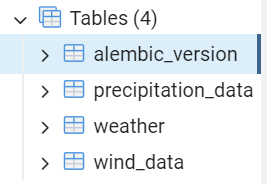
Також між моделями створено двосторонній зв’язок типу «один до одного» (one-to-one), що спрощує навігацію між основними погодними даними та інформацією про вітер і опади. Цей зв’язок реалізовано за допомогою зовнішнього ключа (ForeignKey) у моделях WindData та PrecipitationData, які посилаються на таблицю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

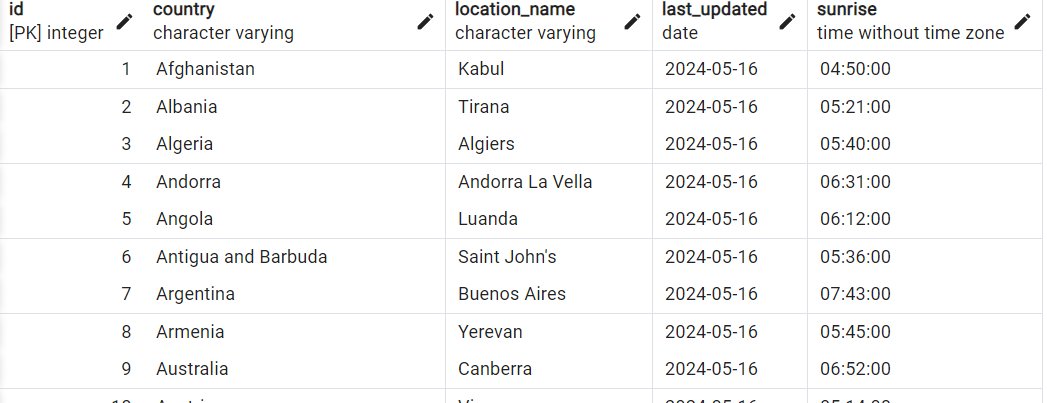
cloud = row["cloud"]

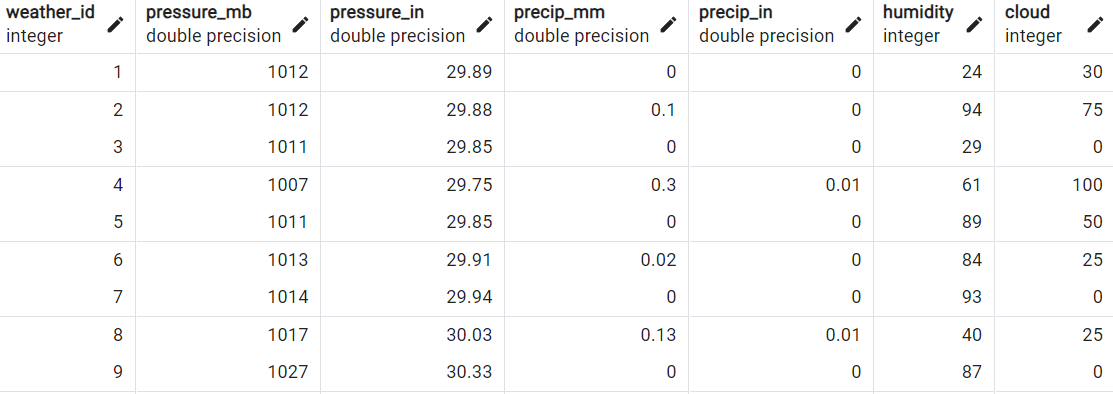
)

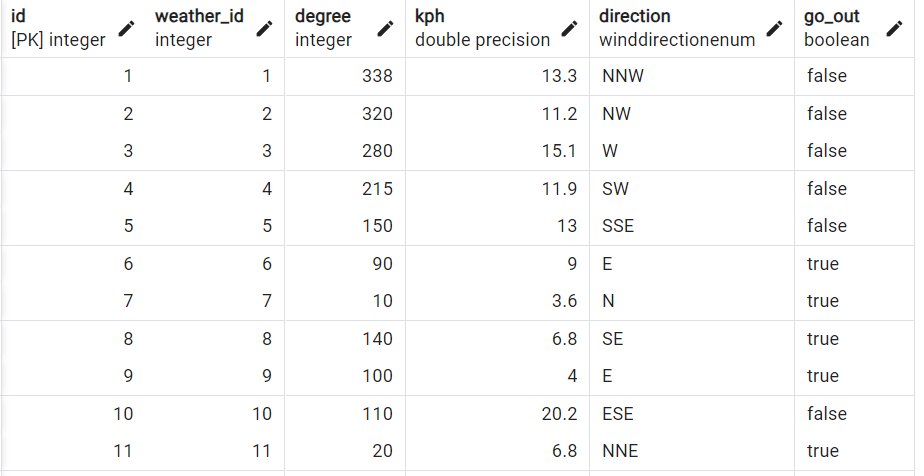
session.add(precipitation)

session.commit()

Дані були успішно перенесено:







2.3 Створення інтерфейсу.

Тепер згідно постаноки 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)

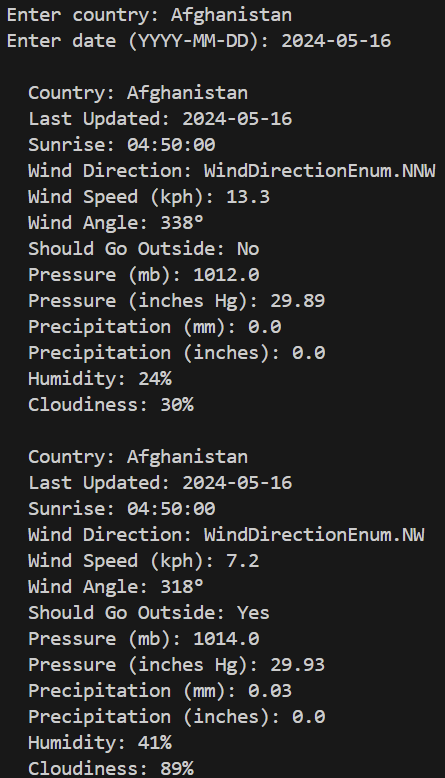
finally:

session.close()

if \_\_name\_\_ == "\_\_main\_\_":

interface()

Результат:



2.4 Міграція з обнієї 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\GlobalWeatherRepository.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()

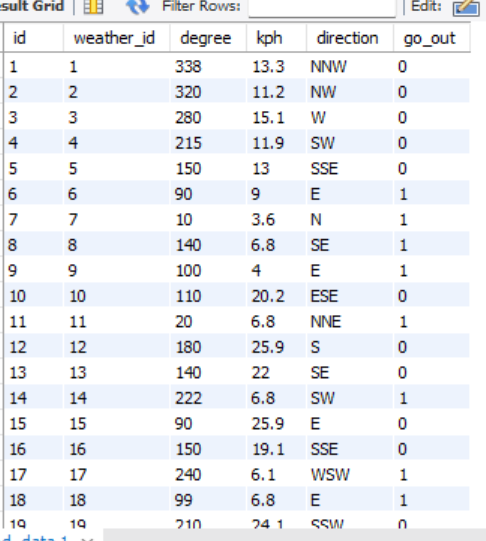
# Insert into MySQL

insert\_data(mysql\_session)

# Insert into PostgreSQL

insert\_data(postgres\_session)

Результат:



**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.