Problem 1 - Data Modeling

Choose a database to use for this coding exercise (SQLite, Postgres, etc.). Design a data model to represent the

weather data records. If you use an ORM, your answer should be in the form of that ORM's data definition format.

If you use pure SQL, your answer should be in the form of DDL statements.

from sqlalchemy import create\_engine, Column, Integer, String

engine = create\_engine('sqlite:///weather\_data.db')

class WeatherData(Base):

\_\_tablename\_\_ = 'weather\_data'

id = Column(Integer, primary\_key=True)

date = Column(String, nullable=False)

max\_temp = Column(Integer, nullable=False)

min\_temp = Column(Integer, nullable=False)

precipitation = Column(Integer, nullable=False)

Write code to ingest the weather data from the raw text files supplied into your database, using the model you designed. Check for duplicates: if your code is run twice, you should not end up with multiple rows with the same data in your database. Your code should also produce log output indicating start and end times and number of records ingested.

import sqlite3

from datetime import datetime

def create\_table(cursor):

cursor.execute('''

CREATE TABLE IF NOT EXISTS weather\_data (

date TEXT PRIMARY KEY,

max\_temp INTEGER NOT NULL,

min\_temp INTEGER NOT NULL,

precipitation INTEGER NOT NULL

)

''')

def insert\_record(cursor, record):

cursor.execute('''

INSERT OR IGNORE INTO weather\_data (date, max\_temp, min\_temp, precipitation)

VALUES (?, ?, ?, ?)

''', record)

def process\_file(file\_path, cursor):

with open(file\_path) as file:

records = [line.strip().split('\t') for line in file]

for record in records:

insert\_record(cursor, record)

def main():

start\_time = datetime.now()

db\_path = 'weather\_data.db'

file\_path = 'weather\_data.txt'

conn = sqlite3.connect(db\_path)

cursor = conn.cursor()

create\_table(cursor)

process\_file(file\_path, cursor)

conn.commit()

conn.close()

end\_time = datetime.now()

duration = end\_time - start\_time

num\_records = len(open(file\_path).readlines())

print(f"Processed {num\_records} records in {duration.total\_seconds()} seconds.")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Problem 3 - Data Analysis**

For every year, for every weather station, calculate:

\* Average maximum temperature (in degrees Celsius)

\* Average minimum temperature (in degrees Celsius)

\* Total accumulated precipitation (in centimeters

Ignore missing data when calculating these statistics.

Design a new data model to store the results. Use NULL for statistics that cannot be calculated.

Your answer should include the new model definition as well as the code used to calculate the new values and store them in the database.

**To store the results of the weather data analysis, we can create a new table in our database. Here's an example of a data model that could be used:**

**CREATE TABLE weather\_stats (**

**year INTEGER NOT NULL,**

**station\_id TEXT NOT NULL,**

**avg\_max\_temp REAL,**

**avg\_min\_temp REAL,**

**total\_precipitation REAL,**

**PRIMARY KEY (year, station\_id)**

**);**

**Here's some Python code that calculates the required statistics and stores them in the weather\_stats table:**

import sqlite3

def calculate\_statistics(cursor):

cursor.execute('SELECT DISTINCT year, station\_id FROM weather\_data')

years\_stations = cursor.fetchall()

for year, station\_id in years\_stations:

cursor.execute('''

SELECT AVG(max\_temp)/10.0, AVG(min\_temp)/10.0, SUM(precipitation)/100.0

FROM weather\_data

WHERE year=? AND station\_id=? AND max\_temp != -9999 AND min\_temp != -9999 AND precipitation != -9999

''', (year, station\_id))

row = cursor.fetchone()

avg\_max\_temp, avg\_min\_temp, total\_precipitation = row

cursor.execute('''

INSERT INTO weather\_stats (year, station\_id, avg\_max\_temp, avg\_min\_temp, total\_precipitation)

VALUES (?, ?, ?, ?, ?)

''', (year, station\_id, avg\_max\_temp, avg\_min\_temp, total\_precipitation))

def main():

db\_path = 'weather\_data.db'

conn = sqlite3.connect(db\_path)

cursor = conn.cursor()

calculate\_statistics(cursor)

conn.commit()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Choose a web framework (e.g. Flask, Django REST Framework). Create a REST API with the following GET endpoints:**

**/api/weather**

**/api/weather/stats**

**Both endpoints should return a JSON-formatted response with a representation of the ingested/calculated data in**

**your database. Allow clients to filter the response by date and station ID (where present) using the query string.**

**Data should be paginated.**

**Include a Swagger/OpenAPI endpoint that provides automatic documentation of your API.**

**Your answer should include all files necessary to run your API locally, along with any unit tests.**

from typing import List, Optional

from fastapi import FastAPI, Query

from pydantic import BaseModel

from sqlalchemy.orm import Session

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import pandas as pd

# Define FastAPI app

app = FastAPI()

# Define SQLAlchemy engine and session factory

SQLALCHEMY\_DATABASE\_URL = "sqlite:///./weather\_data.db"

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Define base class for SQLAlchemy models

Base = declarative\_base()

# Define models

class WeatherData(Base):

\_\_tablename\_\_ = "weather\_data"

id = Column(Integer, primary\_key=True, index=True)

date = Column(String(8), index=True)

station\_id = Column(Integer, index=True)

max\_temp = Column(Float)

min\_temp = Column(Float)

precipitation = Column(Float)

class Station(Base):

\_\_tablename\_\_ = "stations"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, index=True)

# Define Pydantic schemas

class WeatherDataCreate(BaseModel):

date: str

station\_id: Optional[int]

max\_temp: float

min\_temp: float

precipitation: float

class WeatherDataUpdate(BaseModel):

date: Optional[str]

station\_id: Optional[int]

max\_temp: Optional[float]

min\_temp: Optional[float]

precipitation: Optional[float]

class WeatherDataOut(BaseModel):

id: int

date: str

station\_id: Optional[int]

max\_temp: float

min\_temp: float

precipitation: float

class StationOut(BaseModel):

id: int

name: str

# Define CRUD operations for WeatherData

def get\_weather\_data(db: Session, skip: int = 0, limit: int = 100, date: Optional[str] = None, station\_id: Optional[int] = None) -> List[WeatherDataOut]:

weather\_data = db.query(WeatherData)

if date:

weather\_data = weather\_data.filter(WeatherData.date == date)

if station\_id:

weather\_data = weather\_data.filter(WeatherData.station\_id == station\_id)

weather\_data = weather\_data.offset(skip).limit(limit).all()

return weather\_data

def create\_weather\_data(db: Session, weather\_data: WeatherDataCreate) -> WeatherDataOut:

db\_weather\_data = WeatherData(date=weather\_data.date, station\_id=weather\_data.station\_id, max\_temp=weather\_data.max\_temp, min\_temp=weather\_data.min\_temp, precipitation=weather\_data.precipitation)

db.add(db\_weather\_data)

db.commit()

db.refresh(db\_weather\_data)

return db\_weather\_data

def update\_weather\_data(db: Session, weather\_data\_id: int, weather\_data: WeatherDataUpdate) -> WeatherDataOut:

db\_weather\_data = db.query(WeatherData).filter(WeatherData.id == weather\_data\_id).first()

if weather\_data.date:

db\_weather\_data.date = weather\_data.date

if weather\_data.station\_id:

db\_weather\_data.station\_id = weather\_data.station\_id

if weather\_data.max\_temp:

db\_weather\_data.max\_temp