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.

CREATE TABLE weather\_data (

id INTEGER PRIMARY KEY,

date TEXT NOT NULL,

max\_temp INTEGER NOT NULL,

min\_temp INTEGER NOT NULL,

precipitation INTEGER NOT NULL

);

**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.**

import pandas as pd

from sqlalchemy import create\_engine

# Define the SQLite database connection

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

# Load the weather data into a pandas DataFrame

weather\_data = pd.read\_csv('weather\_data.txt', delimiter='\t', header=None,

names=['date', 'max\_temp', 'min\_temp', 'precipitation'])

# Replace missing values (-9999) with NaN

weather\_data = weather\_data.replace(-9999, pd.NA)

# Convert the date column to a datetime data type

weather\_data['date'] = pd.to\_datetime(weather\_data['date'], format='%Y%m%d')

# Add columns for year and station ID

weather\_data['year'] = weather\_data['date'].dt.year

weather\_data['station\_id'] = 'XYZ' # Replace with actual station ID

# Calculate the required statistics using groupby

stats = weather\_data.groupby(['year', 'station\_id']).agg(

avg\_max\_temp=('max\_temp', 'mean'),

avg\_min\_temp=('min\_temp', 'mean'),

total\_precipitation=('precipitation', 'sum')

)

# Convert the temperature statistics from tenths of a degree Celsius to degrees Celsius

stats['avg\_max\_temp'] = stats['avg\_max\_temp'] / 10.0

stats['avg\_min\_temp'] = stats['avg\_min\_temp'] / 10.0

# Insert the statistics into the SQLite database using SQLAlchemy

stats.to\_sql('weather\_stats', engine, if\_exists='replace', index=True)

project

│ app.py

│ requirements.txt

│ swagger.json

│ swagger.yaml

│

└───models

│ │ weather.py

│

└───tests

│ │ test\_api.py

│

└───utils

│ pagination.py

app.py - This is the main Flask application file that defines the REST API endpoints and initializes the Swagger documentation.

from flask import Flask, jsonify, request, url\_for

from flask\_restful import Api, Resource, reqparse

from flask\_swagger\_ui import get\_swaggerui\_blueprint

from models.weather import Weather, WeatherStats

from utils.pagination import paginate\_results

app = Flask(\_\_name\_\_)

api = Api(app)

# Initialize Swagger documentation

SWAGGER\_URL = '/swagger'

API\_URL = '/swagger.json'

swagger\_blueprint = get\_swaggerui\_blueprint(

SWAGGER\_URL,

API\_URL,

config={

'app\_name': 'Weather API'

}

)

app.register\_blueprint(swagger\_blueprint, url\_prefix=SWAGGER\_URL)

# Define request parser for filtering by date and station ID

parser = reqparse.RequestParser()

parser.add\_argument('start\_date', type=str)

parser.add\_argument('end\_date', type=str)

parser.add\_argument('station\_id', type=str)

class WeatherAPI(Resource):

def get(self):

# Parse query string for filters

args = parser.parse\_args()

start\_date = args['start\_date']

end\_date = args['end\_date']

station\_id = args['station\_id']

# Query database for weather data

weather\_data = Weather.query.filter\_by(station\_id=station\_id)

if start\_date:

weather\_data = weather\_data.filter(Weather.date >= start\_date)

if end\_date:

weather\_data = weather\_data.filter(Weather.date <= end\_date)

weather\_data = weather\_data.paginate(page=request.args.get('page', 1, type=int), per\_page=10)

# Convert weather data to JSON and add pagination links

results = []

for weather in weather\_data.items:

weather\_dict = {

'date': str(weather.date),

'max\_temp': weather.max\_temp,

'min\_temp': weather.min\_temp,

'precipitation': weather.precipitation,

'station\_id': weather.station\_id

}

results.append(weather\_dict)

pagination\_links = paginate\_results(weather\_data, 'api.weather')

return jsonify({'results': results, 'pagination': pagination\_links})

api.add\_resource(WeatherAPI, '/api/weather')

class WeatherStatsAPI(Resource):

def get(self):

# Parse query string for filters

args = parser.parse\_args()

start\_date = args['start\_date']

end\_date = args['end\_date']

station\_id = args['station\_id']

# Query database for weather stats

weather\_stats = WeatherStats.query.filter\_by(station\_id=station\_id)

if start\_date:

weather\_stats = weather\_stats.filter(WeatherStats.year >= start\_date[:4])

if end\_date:

weather\_stats = weather\_stats.filter(WeatherStats.year <= end\_date[:4])

weather\_stats = weather\_stats.paginate(page=request.args.get('page', 1, type=int), per\_page=10)

# Convert weather stats to JSON and add pagination links

results = []

for stats in weather\_stats.items:

stats\_dict = {

'year': stats.year,

'station\_id': stats.station\_id,

'avg\_max\_temp':