

Problem 1:

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 extensions import db

class WeatherData(db.Model):
    __tablename__ = 'weather_data'
    id = db.Column(db.Integer, primary_key=True)
    station_id = db.Column(db.String(50), nullable=False)
    date = db.Column(db.Date, nullable=False)
    max_temp = db.Column(db.Integer)
    min_temp = db.Column(db.Integer)
    precipitation = db.Column(db.Integer)
```

Problem 2:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SNOWFLAKE

PS D:\Corteva\corteva\weather-data-pipeline-api\src> python init_db.py
Database tables created
● PS D:\Corteva\corteva\weather-data-pipeline-api\src> python ingest_data.py
INFO: __main__:Data ingestion started at 2024-07-28 20:44:00.520936
INFO: __main__:Data ingestion completed at 2024-07-28 20:55:18.729044
INFO: __main__:Total Duration: 0:11:18.208108
INFO: __main__:Total records ingested: 135808
INFO: __main__:Total duplicate records found: 0
● PS D:\Corteva\corteva\weather-data-pipeline-api\src> python ingest_data.py
INFO: __main__:Data ingestion started at 2024-07-28 20:55:25.965438
INFO: __main__:Data ingestion completed at 2024-07-28 21:10:02.633921
INFO: __main__:Total Duration: 0:14:36.668483
INFO: __main__:Total records ingested: 0
INFO: __main__:Total duplicate records found: 135808
● PS D:\Corteva\corteva\weather-data-pipeline-api\src> python calculate_statistics.py
● Weather statistics calculated and stored successfully
○ PS D:\Corteva\corteva\weather-data-pipeline-api\src> 
```



```

        func.extract('year', WeatherData.date) == year,
        WeatherData.max_temp != None
    ).scalar()

    avg_min_temp = db.session.query(func.avg(WeatherData.min_temp)).filter(
        WeatherData.station_id == station_id,
        func.extract('year', WeatherData.date) == year,
        WeatherData.min_temp != None
    ).scalar()

    total_precipitation =
db.session.query(func.sum(WeatherData.precipitation)).filter(
        WeatherData.station_id == station_id,
        func.extract('year', WeatherData.date) == year,
        WeatherData.precipitation != None
    ).scalar()

    weather_stats = WeatherStatistics(
        station_id=station_id,
        year=year,
        avg_max_temp=avg_max_temp,
        avg_min_temp=avg_min_temp,
        total_precipitation=total_precipitation
    )

    db.session.add(weather_stats)

    db.session.commit()

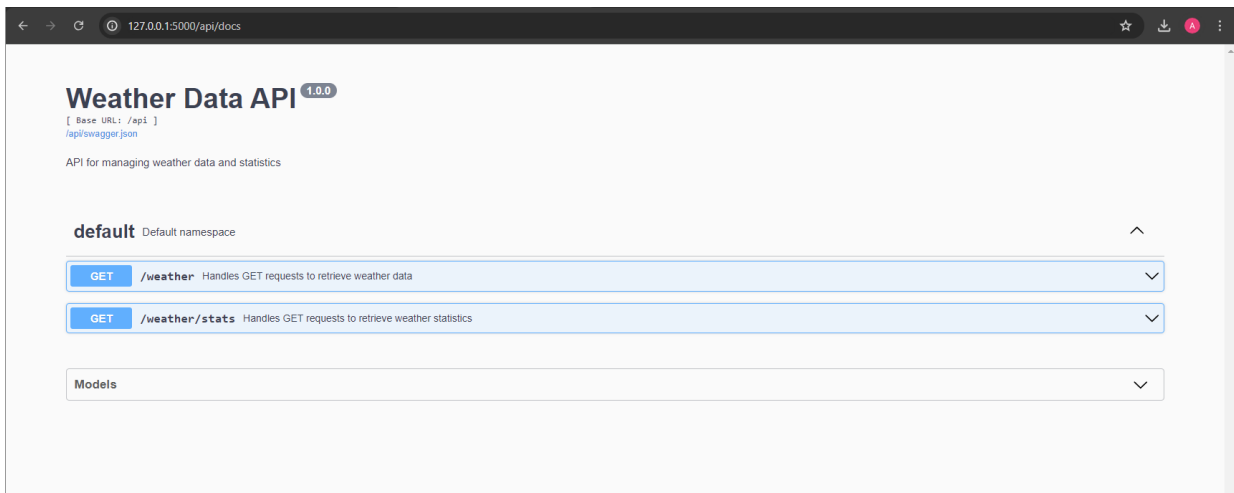
    print("Weather statistics calculated and stored successfully")

if __name__ == '__main__':
    calculate_statistics()

```

Problem 4:

Swagger API Documentation



127.0.0.1:5000/api/weather



127.0.0.1:5000/api/weather/stats



GET endpoint /api/weather:

Clients can filter by date and station ID as shown below.

GET /weather Handles GET requests to retrieve weather data

Filters data by station_id, start_date, and end_date.
Supports pagination.

Parameters Cancel

Name	Description
station_id string (query)	<input type="text" value="USC00113879"/>
start_date string (query)	<input type="text" value="1985-01-01"/>
end_date string (query)	<input type="text" value="1985-01-10"/>
page integer (query)	<input type="text" value="1"/>
per_page integer (query)	<input type="text" value="10"/>
X-Fields string(\$mask) (header)	An optional fields mask <input type="text" value="X-Fields"/>

Execute Clear

Responses Response content type: application/json

Responses Response content type: application/json

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:5000/api/weather?station_id=USC00113879&start_date=1985-01-01&end_date=1985-01-10&page=1&per_page=10' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/api/weather?station_id=USC00113879&start_date=1985-01-01&end_date=1985-01-10&page=1&per_page=10
```

Server response

Code Details

200 Response body

```
[
  {
    "id": 104913,
    "station_id": "USC00113879",
    "date": "1985-01-01",
    "max_temp": 8,
    "min_temp": 3,
    "precipitation": 8
  },
  {
    "id": 104914,
    "station_id": "USC00113879",
    "date": "1985-01-02",
    "max_temp": 3,
    "min_temp": -1,
    "precipitation": 0
  },
  {
    "id": 104915,
    "station_id": "USC00113879",
    "date": "1985-01-03",
    "max_temp": -1,
    "min_temp": -7,
    "precipitation": 0
  },
  {
    "id": 104916,
    "station_id": "USC00113879",
    "date": "1985-01-04",
    "max_temp": -1,
    "min_temp": -7,
    "precipitation": 0
  }
]
```

Download

Response headers

```
connection: close
content-length: 1767
content-type: application/json
date: Mon, 29 Jul 2024 07:02:52 GMT
server: Werkzeug/3.0.3 Python/3.11.9
```

Responses

Code	Description
200	Success

GET endpoint /api/weather/stats:

Clients can filter by year and station ID as shown below.

GET

/weather/stats

Handles GET requests to retrieve weather statistics

⌵

Filters data by station_id and year.
Supports pagination.

Parameters

Cancel

Name	Description
station_id string (query)	<input type="text" value="USC00113879"/>
year integer (query)	<input type="text" value="1990"/>
page integer (query)	<input type="text" value="1"/>
per_page integer (query)	<input type="text" value="10"/>
X-Fields string(\$mask) (header)	An optional fields mask <input type="text" value="X-Fields"/>

Execute

Clear

Responses

Response content type

application/json

⌵

Responses

Response content type

application/json

⌵

Curl

```
curl -X 'GET' \
'http://127.0.0.1:5000/api/weather/stats?station_id=USC00113879&year=1990&page=1&per_page=10' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/api/weather/stats?station_id=USC00113879&year=1990&page=1&per_page=10
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>[{ "id": 26, "station_id": "USC00113879", "year": 1990, "avg_max_temp": 21.248366013071895, "avg_min_temp": 7.6797385620915035, "total_precipitation": 55.7 }, { "id": 408, "station_id": "USC00113879", "year": 1990, "avg_max_temp": 21.248366013071895, "avg_min_temp": 7.6797385620915035, "total_precipitation": 557 }]</pre></div><div><div>Response headers</div><div><pre>connection: close content-length: 433 content-type: application/json date: Mon, 29 Jul 2024 07:04:38 GMT server: Werkzeug/3.0.3 Python/3.11.9</pre></div></div></div>

Responses

Code

Description

200

Success

Extra Credit - Deployment:

To deploy the API, database, and scheduled data ingestion code in the cloud using AWS, we can use the following tools and services:

1. **API Deployment:** We can use AWS Elastic Beanstalk to deploy and manage the Flask API. It handles the infrastructure and scaling automatically.
2. **Database:** For DB, we can use Amazon RDS to set up a managed PostgreSQL database. It handles backups, updates, and scaling.
3. **Scheduled Data Ingestion:** AWS Lambda can be used to run the data ingestion code and we can schedule the Lambda function with Amazon CloudWatch Events to run at specific intervals.

This approach leverages AWS services to manage infrastructure, scalability, and scheduling, allowing you to focus on your application logic.

GitHub Repository:

<https://github.com/Ashray3096/weather-data-pipeline-api/tree/main>