Corteva-Assignment

project name : corteva\_agri app

name : assignment Problem 1 – Data Modeling

#Choose a database to use for this coding exercise (SQLite, Postgres, etc). Design two data models: one to represent the weather data records, and one to represent the yield 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.

SOLUTION: We used SQLite database that is by default provided by Django. We created three Models Weather (in which the column station and date were made unique together), Yield (in which year is taken unique) and AVGWeather (which contains the statistics on weather data)

Problem 2 - Ingestion

Write code to ingest the weather and yield data from the raw text files supplied into your database, using the models 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.

SOLUTION : We created a utils.py which contains three functions ,

Import \_weather: this function looping through all the text files inside the directory wx\_data. Based on current path and reading each file at a time and ingesting in to the database (using weather data model) meanwhile adding file name in station column to add information from which station that data came from.

Import\_yield : this function is also almost same as weather\_import fetching txt file from yld\_data and ingesting into database (using yeld data model).

Get\_all\_data\_info: this functions runs a query on app\_weather table and calculate the result on the given conditions and stores the result in AVGWeather table.

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.

SOLUTION: we used SQLite Query to calculate and aggregate the result and stored the result into AVGWeather Model. This model contains 5 field: station, date, avg\_max\_temp, avg\_min\_temp, and total\_ppt.

def Get\_all\_data\_info():

'''

This function is used to calculate the

\* Average maximum temperature

\* Average minimum temperature

\* Total accumulated precipitation

for every year, for every station and saving into the Database

'''

results = Weather.objects.raw("select id, station,date, avg(max\_temp) as max\_temp,avg(min\_temp),\

sum(ppt) from (select \* from main\_weather\

where max\_temp!=-9999 and min\_temp!=-9999 and ppt!=-9999) t group by station , substring(date,1,4) ")

for res in results:

AVGWeather.objects.get\_or\_create(

station = res.station,

date = res.date,

avg\_max\_temp = res.max\_temp,

avg\_min\_temp =res.min\_temp,

total\_ppt = res.ppt

)

print("query ran succesfully")

Problem 4 - REST API

--------------------

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

/api/weather

/api/yield

/api/weather/stats

Each endpoint should return a JSON-formatted response with a representation of the ingested 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.

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

SOLUTION: Rest API’s is created by using Django Rest Framework, and created serializers.py

to serializer the data models. The API’s are as follows:

<http://127.0.0.1:8000/api/weather> - This API returns all the objects of Weather Model in JSON formatted, users can filter out the response by station ID and date. Pagination is applied to load the data 100 objects per page.

e.g - <http://127.0.0.1:8000/api/weather/?date=20051005&station=USC00111280>

<http://127.0.0.1:8000/api/yield> - This API returns all the objects of Yield Model in JSON formatted. Pagination is applied to load the data 100 objects per page.

<http://127.0.0.1:8000/api/weather/stats/> - This API returns all the objects of AVGWeather Model in JSON formatted, users can filter out the response by station ID and date. Pagination is applied to load the data 100 objects per page.