greatlearning
Power Ahead

Reservoir Data Storage and Retrieval

Introduction

The reservoir data system supports storage of salinity and Calcium concentration (PPM) values from various devices, located at strategic locations within a fresh water reservoir. It also has a device registry.

The data is stored in MongoDB. The focus of the project is to learn some data modelling in MongoDB and to implement various functionalities based on that.

Each device sends either salinity or Calcium PPM data based on its type. There is only one category of user - **admin**.

To some degree, this project is constructed similar to the MongoDB database discussed in the Jupyter notebook for the weekly content on NoSQL. You will find some hints and relevant data modelling suggestions in that notebook.

Housekeeping points

- This is a minimal implementation, and may not follow some standard practices.
- We focus on the main flow, and not much error handling.
- To avoid handling command-line arguments, config file paths are hard-coded in the source files - not a general practice.

Program Organization

The simple program is structured in various layers.

1. **Data model**:
    a. *devices* collection: This stores information about the devices. It has device_id (String), desc (String), type (String - salinity/Calcium) and manufacturer (String) fields. Each device generates only one type of data.
    b. *reservoir_data* collection: This stores the actual sensor data. It contains device_id (String), float (Integer), and timestamp (Date) fields in each document.
2. **src/setup.py**: This program populates the initial data.
    a. First, it drops the database to get rid of any existing data so that it can start afresh.

    b. Then creates a new database called *reservoir_db*
    c. Reads the device information from config/devices.csv and populate those in the *devices* collection
    d. Five salinity and five Calcium PPM sensors are created in the previous step. For sensor data, this creates entries for each device in *reservoir_data* collection. It creates a data entry for each hour from 1st-6th December, 2021, for each device. The temperature and humidity values are randomly generated from a specific range.
3. **src/database.py**: This file hosts the Database class which provides base access to the MongoDB database. It has the connection information stored as static variables. It provides functionality for connecting, fetching, and inserting a document.
4. **src/model.py**: This file hosts DeviceModel, ReservoirDataModel, and DailyReportModel classes.
    a. **DeviceModel**: Provides functionality for finding a device by *device_id* or the auto-generated object id by Mongo. It also supports insertion which ensures that device id remains unique.
    b. **ReservoirDataModel**: Provides functionality for finding a data point by *device_id* and timestamp combination, or by the auto-generated object id by Mongo. It also supports insertion which ensures that data from the same device and the same timestamp is not duplicated.
5. **src/service.py**: This file hosts DeviceService, ReservoirDataService, and DailyReportService classes. Each of these classes forms a service-layer wrapper over the corresponding model classes. This layer acts as an interface between the client code and the data layer code (the model).
6. **src/main.py**: Here, you need to add the functions required for the various problems. Initially, it contains examples of how to connect and use the various models to store and retrieve data.
7. **docs/C02-Project-A-MongoDB-Prep-Material.ipynb**: Contains documentation for various tasks that are helpful in the program.
8. **docs/MongoDB-Install**: Contains pointers to install both MongoDB and MongoDB Compass. Compass is a GUI that can help see and manipulate the various collections and their data.

Problem Statement

The codebase provided along with the problem statement is incomplete. The source code in the file **src/model.py** has gaps in it at certain places, and thus will not produce the result desired in **src/main.py**.

As part of the project, you are required to fill in these gaps with Python code that makes sure the relevant code demonstrates what is required of it. The corresponding client code invoked in **src/main.py** is complete, and will complete the picture.

The problem statement consists of the following steps:

1. (**Easy**)
    a. Add the functionality to query the devices collection - to retrieve a single device entry, by specifying a device id.
    b. Add the functionality to query the reservoir_data collection. You need to support a query that takes a device id and a timestamp as query parameters, and retrieves the desired document, if present, from this collection.

2. (**Medium**) Add reporting support for sensor data. There can be multiple data values inserted in each day for every device. We need to aggregate those to give an overall view for each day, per device.

    We have provided a data model for the *daily_reports* collection. This has support for storing daily aggregated data (**average**, **minimum**, **maximum**) for each device.

    A **data aggregator function** has been provided, which takes existing data from *reservoir_data* collection and adds the aggregated data in *daily_reports* collection. It goes over all the records of *reservoir_data* and aggregate per device per day.

    Normally, this would happen periodically on new data, but here, we mimic it with a bulk aggregation function in this project.

    The client code that invokes this functionality is already present in src/main.py, which is in working order - and will complete the picture.

    a. Complete the implementation of the data retrieval function that takes a **device id** and a **particular day**, and returns **average**, **minimum** and **maximum** data values for that day, fetching data from the *daily_reports* collection.
    b. Complete the implementation of the data retrieval function that takes a **device id** and a **range of day**, and returns **average**, **minimum** and **maximum** data values for each day in the given range, fetching data from the *daily_reports* collection.

3. **(Hard)** Add functionality to detect anomalies in the stored data on salinity and calcium concentration (PPM) values. Use the same daily_reports collection for this purpose.

    The client code that invokes this functionality is already present in src/main.py, which is in working order - and will complete the picture.

    a. Complete the implementation of the function that retrieves the **first anomalous reading of salinity** in the aggregated data over a **range of dates**. This retrieval should stop at the first such detected anomaly.
    b. Complete the implementation of the function that retrieves the **first anomalous reading of calcium** in the aggregated data over a **range of dates**. This retrieval should stop at the first such detected anomaly.

Evaluation Rubric

Total Project Points: **240**

- Basic compilation without errors (**10%**) : **24 Points**
- Correctness:
  Correctness of implementation
  - Problem statement - point 1.a (**20%**) : **48 Points**
  - Problem statement - point 1.b (**20%**) : **48 Points**
  - Problem statement - point 2.a (**20%**) : **48 Points**
  - Problem statement - point 2.b (**20%**) : **48 Points**
  - Problem statement - point 3.a (**5%**) : **12 Points**
  - Problem statement - point 3.b (**5%**) : **12 Points**

Note

- **Minimum Requirements**: The final submission that you upload needs to have a successful compilation, at the least.
  - The basic client code is already present in the main.py file and setup.py file. **Please execute both from the src directory itself, otherwise you'll get access issues to the config files.**
  - To push the data in the MongoDB, execute setup.py. This will create the tables and database in the DB. Once data is available to perform the basic compilation you can execute main.py.
  - This code in main.py is complete, and each piece of client code in it will work when the corresponding gap in src/moel.py is correctly filled up. As you complete the code to fulfil the functionality for parts 1.a, 1.b, 2.a, 2.b, 3.a, and 3.b of the problem statement, also check if the implemented functionality works. Basically, go step by step. That way, you can verify if each step of implementation is working.
  - If you do not attempt any part(s) of the problem statement, make sure that the entire program still gives successful compilation. For this, make sure that the related client code within main.py compiles successfully and it does not give any run-time error. This is to ensure that the implemented parts compile and run as you desire.

- **Program Output Format**:
  - We have added an additional output file in the zip folder. This output file exhibits few of the test cases pertaining to the tasks mentioned in the problem statement.
  - The idea to have this file is to help you understand that after the completion of project how you output on the console should look like.
  - Please understand that this has limited print statements and limited output, if you want to add more or want to change the way data is appearing on the screen. Feel free to do it.

- It is just for representation purposes, your console output can be different from the provided output file.
- Your marks will not be affected by how your data is appearing on the console. We will be looking into the implementation and aggregate data available in MongoDB.
- To access the output file look for **expected_sample_output.txt** file.

Program Instructions

1. Download the zipped folder named **C02-Project-A-Assessment-Reservoir-Data.zip**, unzip it on your local machine, and save it. Go into the directory named **C02-Project-A-Assessment-Reservoir-Data.**

2. Make sure you have Python 3.6 or higher installed. At your command prompt, run:

   ```
   $ python --version
   Python 3.7.3
   ```
   If not installed, install the latest available version of Python 3.

3. Please install **MongoDB** and **MongoDB Compass**. Please see docs/MongoDB-Install for instructions on the download location and setup.

4. Install **pymongo** and **bson** python modules. Please see **docs/C02-Project-A-MongoDB-Prep-Material.ipynb** for help in installing modules

5. Modify the connection parameters in **src/Database.py** if needed, for your setup.

6. Go to the **src** folder, and run **setup.py** to create the database, collections and to populate data in them, using **config/devices.csv**. Please note that this clears the database and adds everything again whenever you run it.

   ```
   $ python3 setup.py (On many Linux platforms)
                          OR
   $ python setup.py      (On Windows platforms)
   ```
   In any case, one of these two commands should work.

7. You can now examine and run **main.py**. This will currently run various simple calls that show how to use the various models. As you solve the problems, you'll be frequently modifying and running this file. You can comment or modify the initial code as needed.

8. Alternatively, you could install a popular Python **IDE**, such as **PyCharm** or **Visual Studio Code**, and select a command to build the project from there.

9. Once the program is ready to submit, zip the parent folder **C02-Project-A-Assessment-Reservoir-Data**, and upload the new zip file as **C02-Project-A-Assessment-Reservoir-Data.zip**.It is now ready for evaluation.