Data Exercise

Task Definition

One of the primary ways the energy industry communicates information is via file flows between the shipper (Us) and the Central Data Service Provider, XOSERVE. The files vary in size from just a few hundered bytes to tens of megabytes of information (For example half-hourly meter reads from smart meters). This excercise presents you with a fictional new file flow that the data science team are looking to use for some of their downstream analytics. It is your task to load this new fileflow based on the format specification and create a database schema of your choosing that will allow downstream analytics to use this information for reporting or predictive modelling.

Submission Guidelines

We would like to be able to run your ingestion script to get an idea of how performant it is and to inspect its behaviour, therefore we ask that you try and keep the tech stack you use as portable as possible. We recommend you use sqlite or postgres as the RDBMS and Python with any libraries that are available on PyPI/conda to insert/parse the data.

1. You may use a data analysis library such as Pandas/anything available on PyPI
2. You should use an ORM to define your models
3. We would expect this dataset to require more than one table to be stored efficiently
4. You may incorporate this functionality into a simple REST API using popular Python web-frameworks such as Flask or Django if you prefer
5. Depending on your experience you may use some kind of containerisation to make it easier for us to deploy/run but this is by no means necessary, a python script/module is fine as well.
6. You should document your entry point's locaton and usage.

Our preferred method of submission is via a public repository on GitHub (Or your preferred source control hosting). This allows us to see how you approach the problem, the more detail you go into your commit messages the better.

File Specification

As part of the smart meter rollout the industry has announced a new fileflow, the SMRT file. This contains hourly reads as they are processed throughout the day. The format of the file is a CSV with a header and footer to ensure the whole file was transferred as partial file transfers do occur ocassionaly.

Record Structure

A typical file flow has the following components:

1. Record Type - Unique identifier for the record
2. Comma delimited fields

The field names are not stored in the file and can be found in reference documentation.

In this case the records have the format:

1. HEADR
    - Record Identifier
    - File type identifier
    - Company ID (Gazprom)
    - File creation date `YYYYMMDD` (UTC)
    - File creation time `HHMMSS` (UTC)
    - File generation number - Matches `(PN|DV)[0-9]{6}` (Production/Dev + File Number)
2. CONSU
    - Record Identifier
    - Meter Number
    - Measurement date (UTC) - `YYYYMMDD`
    - Measurement time (UTC) - `HHMM`
    - Consumption
3. TRAIL
    - Record Identifier

Example File

`...`- *indicates lines are ommitted for space reasons*

```
# File: PN007505.SMRT

"HEADR","SMRT","GAZ","20191011","134942","PN007505"

"CONSU","0000000001","20190928","0000",0.00

"CONSU","0000000001","20190928","0100",1.52

"CONSU","0000000001","20190928","0200",0.73

"CONSU","0000000001","20190928","0300",0.44

...

"CONSU","0000000002","20190928","0000",3.02

"CONSU","0000000002","20190928","0100",4.47

"CONSU","0000000002","20190928","0200",1.23

"CONSU","0000000002","20190928","0300",9.89

...

"TRAIL"
```

The timeseries data is recorded such that each measurement date/time is the start of the measurement period and it is implied that the data is fixed frequency hourly. A gas meter is uniquely identified the meter number. A file can have arbitrary amounts of timeseries data and can start and end partially throughout the day

Requirements

The task is to write a program that will upload these files as they arrive in a given directory to a database. You should record when each file was recieved and which rows came from which file for auditability. The program should reasonably validate the data to ensure that bad data does not go into the database.

1. The program should handle recieving the same datetime for a given meter by overwriting the original value with the new one
2. If the header or footer is not present or in the incorrect format then the file should not be loaded
3. If the same filename has already been loaded it should not be loaded again

The program can implement the data upload either by directly reading the files in a script or if you prefer to use a web framework it can upload the files with a web front end or background service.

If you have time some you could also write an API (REST/SOAP or python module) to get different views of the data. For example:

1. How many meters are in the dataset?
2. What is all the data for a given meter?
3. How many files have we recieved?
4. What was the last file to be recieved?
5. etc...

If any of these requirements are too vague or unclear then please do not hesitate to ask for more information.