Flexion Data Engineering Challenge

The purpose of this challenge is to see what agile engineering looks like to you. We want clean, maintainable, production-quality code that can be built, tested and peer reviewed by any typical engineering team. Because we are focused on multiple dimensions of your solution, the functional requirements are narrow and focused.

We aren't looking for fluency with any particular language or framework but we would like candidates to avoid using Managed Services (such as Glue or Snowflake) which tend to focus more on step-by-step procedures instead of engineered solutions that can exist within a full development lifecycle.

Candidates should focus on proper engineering practices with the purpose of creating a testable, maintainable and easily extendable ETL solution.

After we receive your design, we will schedule a discussion with you to talk about what you did, and why.

The Problem:

You work for an insurance company that is implementing a *new* billing system. While the billing system itself is not fully implemented, you are tasked with the data ingestion process that the billing system will use. We also would like the data stored such that our data science department can perform various analyses on the data at any time and with up to date data as it is available.

Currently, there are 3 feeds that need to be ingested into the system:

- Patient Information - various details about the patient
- Provider Information (Physician / Doctor) - various details about the provider
- Claim Information - various details about the claim needing to be billed

Our goal is to be able to ingest these 3 separate feeds of data into the system so that they can be accessed and queried across each other.

Provided data:

We will provide a CSV file for each of the feed files. Each feed will have a **priming** file which represents the initial state of the data as well as an **update** file that will represent a single update iteration for the data.

To simplify the ingestion rules for the update logic, both the **priming** and the **update** files will follow the exact same structure. Additionally, each update file only represents **New** records for

that data source or **Modified** records for that data source. There is no need to consider that records are deleted in any of the sources.

Below are the details of the data:

Patient Data

1. ***patient_id*** (string) - the patient's identification number
2. ***fname*** (string) - first name
3. ***lname*** string) - last name
4. ***date_of_birth*** (date) - birth date
5. ***address*** (string) - current address
6. ***city*** (string) - city
7. ***state*** (string) - state
8. ***zipcode*** (string) - zipcode
9. ***date_of_residence*** (date) - starting date at this residence
10. ***last_modified_date*** (date) - last time record was modified

Provider Data

1. ***provider_id*** (string) - provider id
2. ***fname*** (string) - first name
3. ***lname*** (string) - last name
4. ***facility_name*** (string) - facility or practice name
5. ***address*** (string) - street address
6. ***city*** (string) - city
7. ***state*** (string) - state
8. ***zipcode*** (string) - zipcode
9. ***specialty_code*** (string) - provider specialty
10. ***start_date*** (date) - date started at facility/practice

    Valid values for ***specialty_code*** are:
    - family_physician
    - orthopedist
    - heart_surgeon
    - physical_therapist

Claim Data

1. ***claim_id*** (string) - claim number
2. ***patient_id*** (string) - patient id
3. ***provider_id*** (string) - provider id
4. ***visit_type*** (string) - type of visit
5. ***total_cost*** (numeric) - cost of visit in dollars and cents (a value of 74.22 equals $74.22)
6. ***coverage_type*** (numeric) - percentage of cost covered
7. ***date_of_service*** (date) - service date

8. ***claim_date*** (date) - start of claim
9. ***billed*** (numeric) - if bill was sent to patient
10. ***last_modified_date*** (date) - last time claim was updated

Valid values for ***visit_type*** are:

- physical
- routine
- illness
- surgery

Valid values for ***coverage_type*** are:

- 100
- 50
- 0

Note about Claims Data
Due to utilizing purely fictional patients, providers and ancillary random data associated with them, the **priming** claims data provided can (and most likely will) contain multiple claims for a single patient with overlapping dates and disparate provider values. Assume that the data in the claims **priming** file is accurate irregardless of all the claims for any single patient.

Requirements:

The requirements below are not intended to suggest any particular solution but to guide you in your approach to solving the problem.

1. Our data scientists must be able to query the data based on the details of the originating source schema's. You can use whatever AWS services you like. You may use the free tier so as to not get charged.
2. The system should accommodate that fields are verified correctly and verification issues can be reviewed.
3. We want clean, maintainable, production-quality code.

Example scenarios (not exhaustive):

- Get all claims that have not been billed. The results should include the claim information as well as the patient's name, patient's address and provider's name.
- Get all providers who have had claims in the last 30 days where the insurance claim must be 100% covered.
- Get all patients who have had a surgery claim with a specific provider id.

Submitting your response:

We understand that you probably have a lot going on. So please make sure you negotiate enough time in order to produce a solution you would be proud of. You will not be penalized for taking the time you need, just keep in mind that we are continuing to screen candidates for the position you are applying for.

1. Create a private GitHub or GitLab repo that will contain your code. Share it with the GitHub/GitLab user named "`FlexionCodeReview`" giving that user permission to at least read your repo.
2. Include a `README.md` that explains how to install, build, test, deploy and run (or access) your program.
3. In your `README.md`, add a prioritized list of five or more development tasks you would do next to improve your solution to the code challenge.
4. Notify your technical recruiter that you are done, and provide your repo URL.
5. No more changes can be committed after the deadline negotiated between you and our technical recruiter.