**Take‑Home Assignment: EMR Data Pipeline**

**Estimated Time Budget**: *≈ 4 hours(give me a heads up if you need extra time)*

**🎯 Background & Context**

We help home‑care agencies automate and streamline their operations—saving admin hours, boosting caregiver satisfaction, and improving patient outcomes. A critical part of this mission is ingesting, normalizing, and analyzing data from many Electronic Medical Record (EMR) systems, each with its own quirks.

You’ll work with two anonymized CSV exports from a home‑care EMR:

|  |
| --- |
|  |
| caregivers.csv |
| carelogs.csv |
|  |

Your goal is to clearly demonstrate your data engineering and analytical capabilities by transforming and structuring this data for clear analysis.

**🚀 Assignment Overview**

**Step 1 – ETL Pipeline**

* Build a simple **ETL pipeline** to load the provided CSV data into PostgreSQL with TypeScript.
* Clearly document your ingestion and transformation logic.

Github: <https://github.com/chan-web-source/zingage-caregivers>

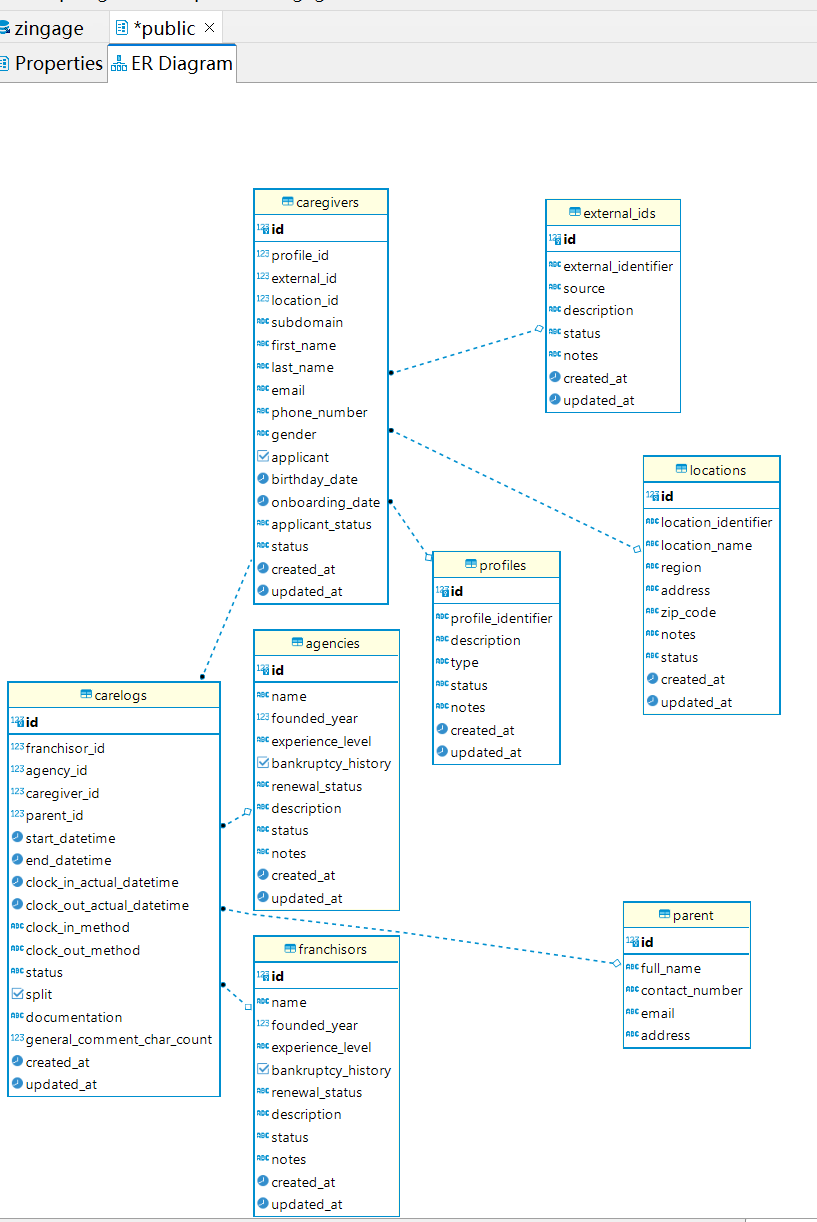
Transformation logic:

1. I started by reviewing both CSV files—caregivers and carelogs—to understand the structure and relationship between them. It was clear that carelogs are work records tied to individual caregivers, making it a child table with caregiver\_id as a foreign key.I then found the connections that carelog is the child table reliant on caregiver as the parent table. Carelog is the work summary (e.g. time, work, location) from the caregiver.
2. I identified all xxx\_id fields as potential references to other tables. I then designed the database starting from parent tables like caregivers, franchisors, and agencies, and worked down to carelogs, ensuring top down schema relationships.
3. I then create a schema, set up tables according to the schema (from parent table to child table) in my local PostgreSQL

**Step 2 – Schema Design & Normalization**

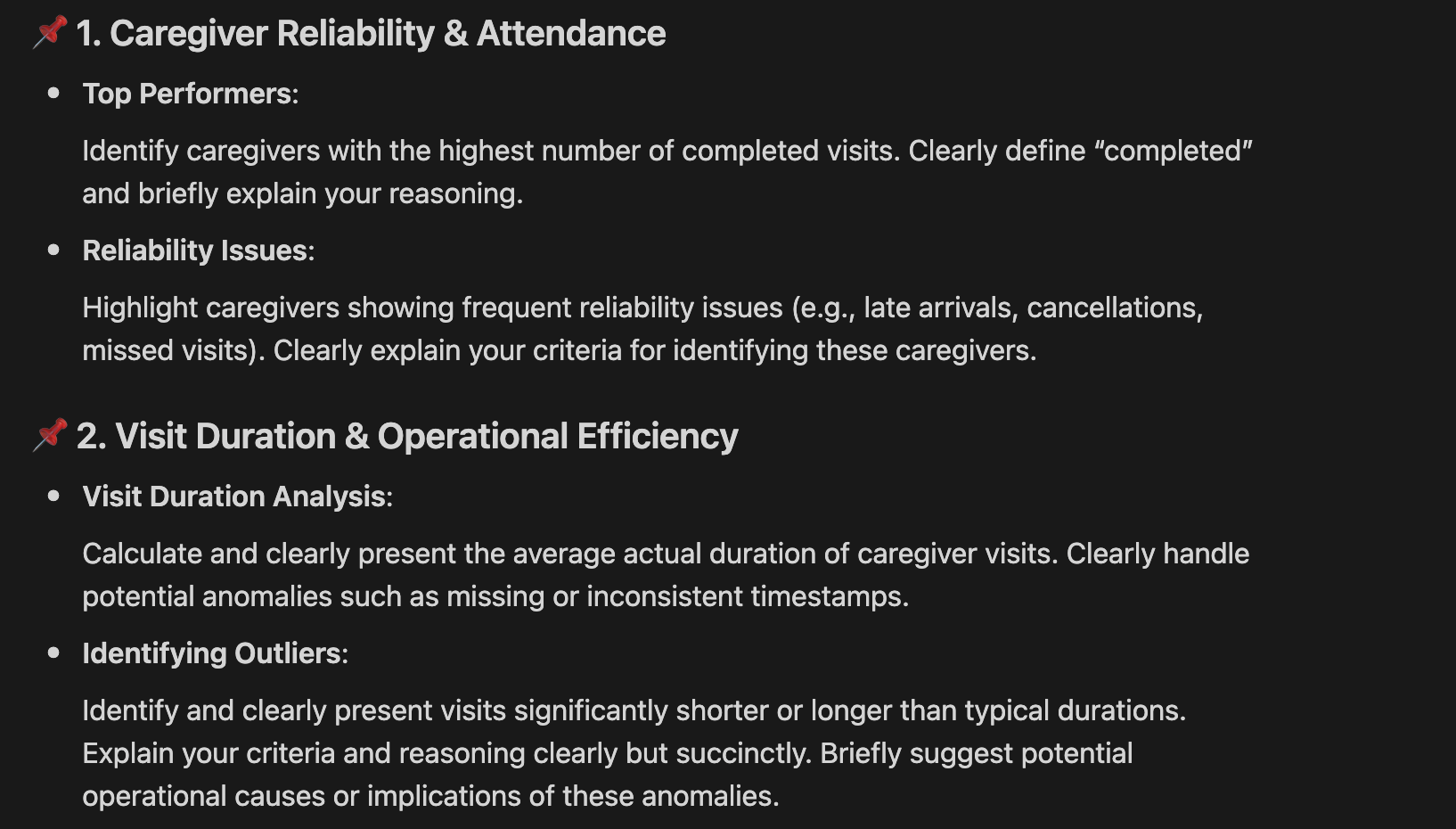
* Normalize and clearly structure your data schema for ease of analytical querying.
* Provide a **schema diagram** (recommended) or clearly written **SQL schema definitions**.
* Clearly document how the two CSV files link together (**Hint:** via caregiver\_id).

Schema design:



1. The carelogs table has a column named **caregiver\_id**, which is a **foreign key referencing the id column** in the caregivers table.
2. both tables **depend on multiple shared** and shares key structural tables with it (franchisors, agencies) with reusable data to avoid inserting same data for both tables

**Step 3: SQL Queries & Analytical Answers**

Write clear SQL queries that answer these business-critical questions. Provide the queries clearly along with formatted example outputs (screenshots or tables). We intentionally leave some aspects open-ended, as we’re interested in how you independently define analytical criteria, handle ambiguity, and approach real-world complexity  
.

* Top performers rank sample outputs:

A screenshot of a computer

AI-generated content may be incorrect.

* Highest numbers of completed visits logic is calculated with
  + Total times in carelogs
  + Reliant on clock\_in\_actual\_datetime, clock\_out\_actual\_datetime
* Status should be type according to number to define if is cancelled
  + Group status type of cancel, get the highest numbers of cancel and rank them in top 10 order
* I saw that some timestamp is shorter some is longer (e.g. next day)
  + It should be reliant also on clock\_in\_method and clock\_out\_method
  + If clock\_out\_method is undefined, the actual work time need to add extra handling or make it not counted
  + Suggestion for enhancement: using linux timestamp over actual dateTime can avoid time range difference (e.g. now is using 2026/4/9 21:03:00 and change to linux timestamp of 1775954580)

A screenshot of a black and white screen

AI-generated content may be incorrect.

* Take “general\_comment\_char\_count “ column to count their overall comments amount.

Filters caregivers who write available comments with in **and group them according to total** comments as percentage of their care logs

1. Firstly , define Standard work limit (40 hours per week )

Write function to calculate Weekly Work Hours per Caregiver (select from carelogs, use clock\_out\_actual\_datetime deduct clock\_in\_actual\_datetime to get the hours) and sum them up

Potential correlation for overtime:

Agency type

Location (time of transportation)

Franchisor experience\_level

clock\_in\_method

These factors can be taken in to account

**Step 4 – AI Usage Guidelines**

* **AI assistance is allowed** (e.g., Copilot/ChatGPT) for brainstorming or boilerplate—but **don’t outsource the whole solution.**
* Create ai\_prompts.txt (or .md) that lists any prompts or instructions you fed to AI and the section of code/output they influenced.
* We reserve the right to dig into any area that looks 100 % AI‑generated.

**1**. Generate Interface & DB Schema from CSV

* Use AI to convert .csv column headers into:
  + TypeScript interfaces
  + PostgreSQL table schemas
* Example instruction pattern:
  + "xxx\_id" → denotes a foreign key reference

2. Scale Functionalities with AI Assistance

* Provide:
  + Previously built functionalities as references (including interface and format)
  + Naming conventions for models, functions, and APIs
* AI helps extend new features with consistent logic and naming

3. Ensure Consistency & Type-Safety

* Validate:
  + Code formatting
  + Strong TypeScript interfaces
* Use AI to cross-check structure and types

4. Documentation Writing

* Bullet-point initial thoughts
* Use AI to rewrite and format them clearly into professional documentation

5. PostgreSQL Query Authoring

* Use AI to:
  + Write queries in a clean, consistent format
  + Ensure readability and structure
* Example: align clauses (e.g., SELECT, FROM, JOIN) and indentation

**Step 5 – README & Assumptions**

Briefly cover:

* Assumptions about ambiguous / missing data.
* Rationale for schema choices & normal forms.
* Key trade‑offs or alternative designs you considered.
* Link to your Loom demo (see below).

**🎥 Loom Walk‑Through (Required)**

Record a **10 minute Loom** (or comparable) video and include it **with your initial submission**. Please cover:

* **ETL approach**: how you ingest and transform the two CSVs.
* **Schema overview**: tables/keys and how caregiver\_id links datasets.
* **One representative query**: run it and briefly explain the result.
* **AI usage call‑out**: where AI assisted (refer to ai\_prompts.txt).

Include the Loom link in your **README** and in your submission email.

**📦 Submission Checklist**

Send either a **public GitHub link** or a **ZIP** named emr\_pipeline\_<yourname>.zip that includes:

* **ETL code (TypeScript)**
* **Database schema**: ERD image *or* schema.sql DDL
* **SQL queries** + **sample outputs** (screenshots or snippets)
* **ai\_prompts.txt** documenting any AI prompts used and what they influenced
* **README** with assumptions, schema rationale, brief run instructions, and the **Loom link**

Email the repo/ZIP **and** Loom link to **hanyao.li@bravozoom.io** with subject **“EMR Pipeline Take‑Home – <Your Name>”.**

**📅 Follow‑Up Interview**

After submitting, your work will be evaluated. If your submitted work is strong, we'll invite you to an remote interview session. During this session:

* You’ll have **~45 minutes** to clearly walk through your solution.
* Clearly explain your thought process, decisions, and results.
* Expect questions from the engineering team about your approach, implementation details, schema choices, SQL logic, and handling of data ambiguity.

**🔎 Evaluation Criteria**

* **Correctness & clarity** of ETL and SQL
* **Data model quality** and clear linkage between files
* **Edge‑case handling** (e.g., missing/invalid timestamps)
* **Communication** (README + concise Loom walkthrough)
* **Appropriate AI usage** (documented in ai\_prompts.txt; demonstrates your understanding)
* **Overall polish** around the ≈4‑hour time budget