Data Engineering Challenge- From Matches to Metrics

Nordeus - Job Fair 2024

About the challenge 🏆

After weeks of intense competition and excitement as players battled for victory, the final whistle blows, leaving us with valuable data from the past season of gameplay (07. Oct. - 03. Nov.). It's time to tackle the data and help us analyze it efficiently!

You are given as input a dataset (JSONL format) where each line is a valid JSON object that represents an event generated by a football manager game. Each event emitted from the game has several fields/parameters that describe the action happening inside the game.

Data description 📊

Dataset (events.jsonl)

Parameter name	Parameter type	Parameter description
event_id	INT	Unique identifier representing an event
event_timestamp	INT	Time of event represented as Unix time
event_type	STRING	One of the following: registration, session_ping, match
event_data	JSON OBJECT	JSON object containing all event-specific data (check event data below)

For our analysis, we are interested in three types of events: registration, session_ping, and match.

- Registration

Registration event is generated when a user registers. At that time, the user gets a unique identifier that we use for all other events, as a way to keep track of him. This event also contains the country that the user came from and the OS of the device that he uses.

Parameter name	Parameter type	Parameter description
country	STRING	Country that the user comes from
user_id	STRING	Unique identifier representing a user
device_os	STRING	OS of the device user registered from. Valid values are iOS, Android, and Web

- Session_ping

A session is the continuous period during which a player interacts with a game. Think of a user session in a game like one 'session' of playing.

We track sessions using session_ping events. This event is being sent when a user opens the game, and every 60 seconds after that, as long as the user has the game open. If more than 60 seconds pass between two ping events, that means they belong to different sessions. First event within a session has type session_start, while the last event has type session_end. These two types are the only ones you need to process if you are using the column “type”.

Parameter name	Parameter type	Parameter description
user_id	STRING	Unique identifier representing a user
type	STRING	Possible values are: session_start, session_end, and “”

Bonus points if you manage to correctly process session data and implement API without using the column “type” (hint: window functions).

- Match

This event is being sent when a match starts and finishes. If the match started, values for home and away goals scored will be NULL. If the event is for match finished, it will have non-NULL values.

Parameter name	Parameter type	Parameter description
match_id	STRING	Identifier representing one match (has the same value for both match start and match end)
home_user_id	STRING	Unique identifier representing a user
away_user_id	STRING	Unique identifier representing a user
home_goals_scored	INTEGER	Amount of goals scored in a match or NULL.
away_goals_scored	INTEGER	Amount of goals scored in a match or NULL.

Dataset (timezones.jsonl)

For simplicity every country will be assigned exactly 1 timezone.

Parameter name	Parameter type	Parameter description
country	STRING	Country code
timezone	STRING	Timezone in the country. Values: “America/New_York”, “Asia/Tokyo”, “Europe/Berlin”, “Europe/Rome”

Tasks 📋

Data cleaning 🧹

- Discard duplicate events (event_id uniquely identifies an event)
- Discard invalid events. Events are considered invalid if they do not meet the requirements specified above.
- Use common sense to discard other events that don’t make any sense

API 💻

- Get user-level stats:
    - Input:
        - user_id (required): represents a unique identifier for a user, described above.
        - date in format YYYY-MM-DD (optional): If no date is specified, then calculate all-time stats for the specified user. If it is specified, then calculate user stats for the date specified.
    - Output:
        - Country of the user
        - Registration datetime in local timezone
        - How many days have passed since this user last logged in. If no date is specified, calculate how many days have between the last login
        - Number of sessions for that user on that day. If no date is provided give the all time number of sessions
        - Time spent in game in seconds for that day or all time if no date is provided
        - Total points won on matches that started on the specified date grouped by whether the user was home or away. If no date is provided give the all time number of points (Win - 3 pts, Draw - 1pt, Loss - 0pt)
        - Bonus: Match time as a percentage of total game time. Represents how much of a player's overall game session was spent actively participating in a match, compared to the entire time spent in the game. If no date is provided give the all time percentage.

- Get game-level stats:
  - Input:
    - date in format YYYY-MM-DD (Optional). If no date is specified, then calculate all-time stats. If it is specified, then calculate stats for that day only.
  - Output:
    - Number of daily active users. If a user has had a session that day, he is considered a daily active user. If no date is specified, then this column represents the number of all users that had at least one session.
    - Number of sessions
    - Average number of sessions for users that have at least one session
    - The user/users with the most points overall

Expectations 🎯

You have been given basic tests in Python that check the validity of your code on events_test.jsonl dataset. The tests given won't be used when grading the project, they are there to help you. **Note**: these tests cover only edge cases that are found in events_test.jsonl, which is only a portion of all edge cases found in events.jsonl.

You are free to completely ignore this skeleton and tests, as well as choose any language of your choice, it won't affect your score.

The input file should be processed **only once** and the target data model should be somehow persisted (in-memory, file, database, etc.) so that queries can be executed efficiently. It is not recommended to process the input file each time a query is executed.

You have the freedom to implement the API in any way you find appropriate and in any language (Python, Java, SQL, etc.) for example:

- REST API implemented in any language
- CLI APP implemented in any language
- Other

One important thing here is that you should add a **documentation** file (possibly README.md). This document should explain your chosen approach for the project and provide clear, user-friendly instructions on how to use the application, including how to start it and install any required dependencies.

Bonus points 💎

- Implementing a detailed and careful data-cleaning process
- Storing the target data model in a SQL database and using SQL to query data
- Implementing the solution as a well-documented REST API
- Processing session data without the column "type"
- Calculating match time as a percentage of total game time

Submission 📤

- Upload the solution to a new public repository in your **GitHub/Gitlab/Bitbucket profile**
- Send an email to jobfair@nordeus.com with a link to your repository with the subject **Data Engineering Challenge** and please add your full name in the email. 🙂

General advice 💡

- Rely on your best judgment!
- If you're stuck, feel free to ask for clarification or help by emailing us at jobfair@nordeus.com.
- If something in the problem seems unclear, state your assumptions explicitly.
- We encourage you to try out the challenge! It's better to send an incomplete solution than not to send one at all! 🙂

The challenge is open until November 17, 2024, end of the day. Good luck!