Solution for Dangineering Technical Assignment

By: Noud Kuilder
At: 22-10-2020

Overview:

Due to its general popularity, open-source, and ease to setup I chose MySQL as the database to use. For production purposes the ease of maintenance, security and scalability to >1M rows makes it well suited for this job. Python is used to submit SQL commands and perform data transformation.

Having had no previous experience with Docker, I found it challenging to set up a DBMS with Python in a way that shows the result of my work in the database. It did result in me being motivated to learn Docker.

I did come up and implemented a working solution for both questions, and I believe these are the most important parts for the open position of Analytics / Data Engineer. Therefore, I hope you can judge my data engineering capabilities without the use of Docker. For question 1 I added a script that exports the data from the local MySQL instance and saves the results as CSVs. Have a look at the code and judge the working of it by the supplied output as CSV.

For question 2, I made a stand-alone Python script that run the SQL queries on Pandas' data frames.

Four Python scripts and one SQL script are included:

- create_tables_scheme_query.sql – SQL statement for all schema and all tables creation
- create_tables_and_schemes.py – execute the SQL commands for schema and table creation
- question1_etl.py – Solution of ETL process for question 1, populates the tables created before
- mysql_to_csv.py – Creates CSVs bearing the names of the tables as present in MySQL Server
- question2_availability.py – Solution to question 2

Besides the scripts the following files are present:

- Event_log.csv – Raw data for assignment
- Services_info.csv – Dimensional table for services
- Account_status_events.csv – Events related to the status of professional's accounts
- Proposal_events.csv – Events related to proposals
- Availability_snapshot.csv – Number of professional's accounts able to propose by day
- requirements.txt – dependencies for stand-alone script question2_availability.py

Question 1:

*"Choose a **relational DB** and **create a data pipeline** to load and transform the dataset. The tables should be optimised for readability and ease of querying by product analysts."*

I normalised the service data in dimensional table. I split the events in two kinds, account status and proposal events to cluster related events, and in such a way that all cells have no NULL or NAN values, specifically the meta_data are only present for proposal events. All table column properties are set to NO NULL thereby performing some data validation.

Overview of the tables:

Table Name	**Description** (primary key in bold, foreign key in blue)	**Explanation**
Services_info	[**service_id**, name_nl, name_en] - Dimensional table	Service ID mapping with human readable (e.g. no hyphen) explanation in both English as Dutch,
Proposal_events	[**event_id**, professional_id_anonymized, service_id, lead_fee, time_stamp]	All proposal related information in one table
Account_status_events	[**event_id**, event_type, professional_id_anonymized, time_stamp]	Table to follow account status

Remarks:

Dimensional modelling dictates a strict separation of Dimensional data and Facts. The ETL largely follows this principle but for the event_types it was chosen not to do so. This is due to the low cardinality, static nature of these events, and for increased readability for data analyses.

The professional_id_anonymized is labelled as a foreign key. I assume there exist a table with, anonymized but more descriptive info about the professionals, like location and profession.

Question 2:

Problem:

"Create an *availability_snapshot* table that would store the amount of active professionals per day."

Specifications:

- An event happening on a day defines the status for this professional for the whole day
    - **Assumption**: multiple events happening on the same day -> last event defines the status of the professional for the whole day

Edge cases:

1. Multiple entries on a single day for the same professional.
2. Repeating of the same action, e.g. 'became_able_to_propose' for a professional already available.

Example, professional_id 244 (top block edge case 2, lower block edge case 1):

	event_id	event_type	professional_id_anonymized	created_at	meta_data
3124	1625	became_able_to_propose	244	2020-01-07 15:46:00	NaN
2418	2376	became_unable_to_propose	244	2020-01-16 08:10:19	NaN
859	1934	became_able_to_propose	244	2020-01-21 12:38:53	NaN
3109	2252	became_able_to_propose	244	2020-02-01 13:28:03	NaN
2426	2504	became_unable_to_propose	244	2020-02-27 10:40:21	NaN
3115	2348	became_able_to_propose	244	2020-02-27 12:38:31	NaN
1037	2508	became_unable_to_propose	244	2020-02-27 12:39:13	NaN

Approach:

Intuitively a cumulative count of 'became_able_to_propose' minus 'became_unable_to_propose' by date would result in the number of available professionals. Edge cases undermine this approach and a more complex solution is required for an accurate answer.

I decided to first create a clean dataset with a row for every interval with a start time and end time. For professionals still active, a future dummy end_date is added. An example of the cleaned interval table is shown below:

professional id	start_date	end_date
1	01/03/2020	05/03/2020
2	01/03/2020	06/03/2020
2	08/03/2020	15/03/2020
3	01/03/2020	01/01/9999

With help of date scaffolding the final solution is obtained.

The operations are done via SQL queries run on Pandas' dataframes with the help of the pandasql library.

Remarks

For most analytical purposes the value derived from the cumulative count approach would be sufficient. Complexity comes at a cost and it would be worth considering taking a simpler but not fully accurate approach over the complex more accurate approach.

Possible performance increase**:** date scaffolding can be done more efficient, see: https://discourse.looker.com/t/sql-pattern-summarizing-entities-with-a-start-end-date-over-time/4868