Big Data IU-S25 Assignment 2

Nikita Yaneev
Innopolis University
Innopolis, Russia
n.yaneev@innopolis.university

I. ABSRACT

This report provides a summary of the database design and data processing work carried out using PostgreSQL, MongoDB, and Neo4j. The project involved designing efficient data models for each database system, considering their specific use cases and strengths. Additionally, data analysis and preprocessing steps were conducted to ensure the quality and integrity of the data before storage. The report outlines the methodologies used, key challenges encountered, and the solutions implemented to optimize data handling across different database technologies.

II. DATA MODELINGL

A. Data preprocessing

Before building the models, my decision was to process the data first. The csv files contained data that could contain missing values or columns that would not help in any way for analysis.

campaings.csv:

For this file, the first thing I did was process the data and convert the numeric values from the string type to numeric form. Also, the 'ab_test' column resulted in the bool type and filled in the gaps with the False value. Because if this information is not available, then in my opinion, this information cannot be considered True. I did not notice any columns that could be deleted, I believe that each column can affect the buyer.

messages.csv

First, you need to convert the temporary data to the datetime format, and process all fields where the True/False value is assumed.

friends.csv

Here it is enough to reduce the data to a numerical form

client_first_purchase_date.csv: Time processing is required here

events.csv: For the data in PostgreSQL, I moved the products data to a separate DataFrame for further interaction And deleted all the omissions.

I processed the data for MongoDB separately, as the structure I wanted is radically different from PostgreSQL and Neo4j. For MongoDB, I processed the data and generated JSON files for them, so I got a convenient structure for importing data into MongoDB. I also ran into difficulties that MongoDB incorrectly processes data from JSON, so I changed the date in the JSON files themselves to get the correct date.

B. PostgreSQL

Data Modeling For database modeling, I used Hackolade, a tool that provides an intuitive interface for designing database schemas and generating code for various database. This tool allows flexible schema customization.

The first database model was designed for PostgreSQL, a relational database that efficiently defines relationships between fields to avoid data duplication. Figure 1 presents the database schema created based on data analysis.

Data Analysis In PostgreSQL, a crucial aspect of database design is defining keys that facilitate efficient searches for unique records across multiple tables.

In the campaignsċsv file, the composite key consists of the fields 'id' (Campaign ID) and 'campaign_type' (Campaign type: bulk, trigger, transactional). In the messages.csv file, the primary key is the field 'id'. To optimize data storage and reduce redundancy, I decided to extract product-related information from eventsċsv into a separate table. This approach ensures that only records containing meaningful values are stored.

Additionally, some fields in the dataset contained missing values. For example, in certain records, 'brand' and 'category_code' were missing, while 'product_id' and 'category_id' were available. To maintain data consistency, only unique product records with known product_id and category_id were retained.

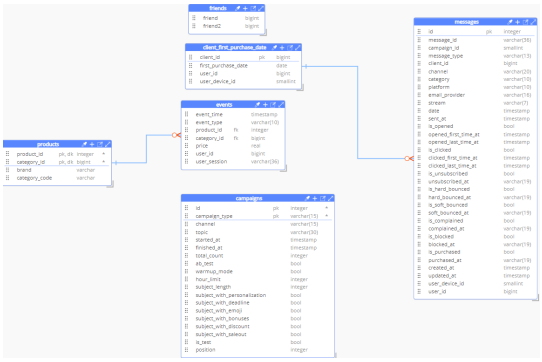

Fig. 1. PostgreSQL - model

C. MongoDB

The second database model was designed for MongoDB, a NoSQL database that provides flexible and scalable data storage. One of MongoDB's key advantages is its ability

to store data in JSON-like documents, allowing for nested structures that simplify data retrieval and reduce the need for complex joins.

Taking advantage of MongoDB's ability to store nested documents, I made the following optimizations:

In the messages collection, all parameters related to messages sent to users were grouped into a separate embedded document called subject. Similarly, all attributes related to user responses were stored in a separate status document. This structure enhances data organization and retrieval efficiency. In the events collection, I separated product- and user-related information into distinct product and user documents. Given MongoDB's ability to store complex objects in a JSON format, this approach ensures better data organization while maintaining flexibility.

In Figure 2, you can see my structure for MongoDB.

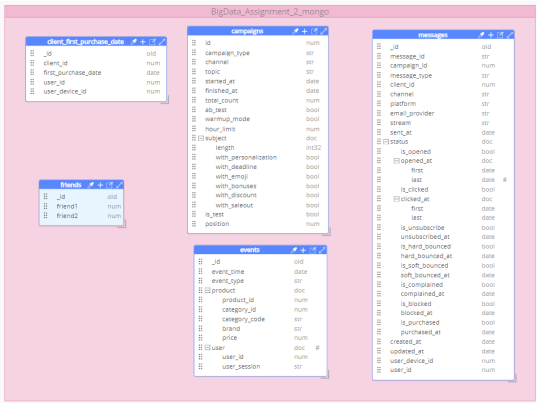

Fig. 2. MongoDB - model

D. Neo4j

The third database model was built using Neo4j, a graph database designed to establish and efficiently traverse relationships between data entities. In Neo4j, each entity is represented as a node, and relationships between them are defined by edges (or relationships) with specific properties.

In my structure, I decided to build a chain of dependencies. For example, the node from which we start filling is the data from campaign.csv linked to messages.csv, messages.csv is linked to a client, where client.csv is linked to itself by user_id, from friends.csv, this is necessary to build dependencies between users and to further analyze the influence of friends on the purchase of goods. At the same time, the client is linked to events.csv, since the user either made a purchase or not, event.csv, in turn, is linked to product.csv in order to understand which product the user bought. An important note is that in Neo4j, it is possible to specify the direction of connections. I decided that there would be bidirectional communication everywhere. My decision is based on the fact that if I know the product that the user bought, then I need to know which campaign sent him the message.

In Figure 3, you can see my structure for Neo4j.

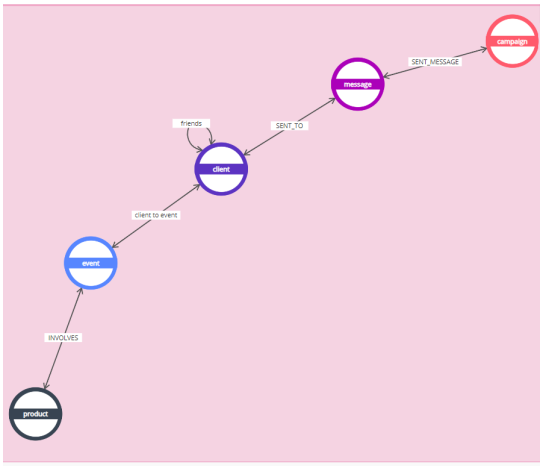

Fig. 3. Neo4j - model

III. Discussion

The results of the data import can be found in the output folder.

In order to discuss the results, they must be. I didn't have time to make requests for data analysis. This is because the data files were large, which meant you had to spend a lot of time waiting for something to happen, and if an error occurred, then everything starts over. Key issues:

1) The size of datasets – for educational purposes, I don't see the difference between 3 million message records and 1 million records.

2) JSON, for the correct processing of time, I had to wait more than an hour for the files to be assembled in JSON format, after it turned out that I also needed to change the date in JSON itself, because of this it took a long time.

3) Neo4j – due to the size of the data, we had to wait an inadequate amount of time. Go to the Neo4j config file and change the memory settings there while the program is running

Collectively, most of the time was spent waiting and correcting mistakes that were not obvious.

Acknowledgment

The style of this report is inspired by the [1].

References

[1] Serhat Uzunbayir. Relational database and nosql inspections using mongodb and neo4j on a big data application. In 2022 7th International Conference on Computer Science and Engineering (UBMK), pages 148–153, 2022. doi:10.1109/UBMK55850.2022.9919589.