Big Data IU-S25 Assignment 2

Vsevolod Klyushev
Innopolis University
Innopolis, Russia
v.klyushev@innopolis.university

***Abstract*—In this assignment, we were required to provide an e-commerce solution for a company to store, receive, and analyze information more efficiently. The main objectives were to implement SQL and noSQL data models for an e-commerce company and analyze its performance. I have developed data storage models for databases such as PostgreSQL, MongoDB, and Neo4j and wrote scripts to migrate from the current solution (storing information in csv files) to them.**

I. Data Modeling

A. Data analysis

The first step to develop a solution is to understand the structure of the data with which we are working. In total, we have 5 files with information about companies, messages, users, events, and connections between users. Based on the information from these tables, I decided to factor out new data objects and remove/reallocate several features. Let us consider them one by one:

1) Messages: This table initially has 34 features, however, not all of them are necessary connected to them. I took the following actions to improve it:

- I dropped column `category`, since it contained only null values and we had no information about its meaning in the assignment description.
- I dropped column `id`, since we already have uuid in column `message_id`.
- `user_id`, `user_device_id` and `email_provider` depends on `client_id` and should be moved from `messages` table to `clients` (also known as `client_first_purchase_date` table)

2) Events: This is the second table that requires splitting. Feature `brand` solely depends on `product_id` and feature `category_code` on `price`. With this information, we can create two more tables `products` and `categories` that would store such dependencies and allow us to remove two unnecessary columns from the table `events`.

Such actions for both `messages` and `events` initial tables helped us to reduce the amount of stored information.

B. Data preprocessing

The provided data was in very bad shape and required a number of actions to make it better:

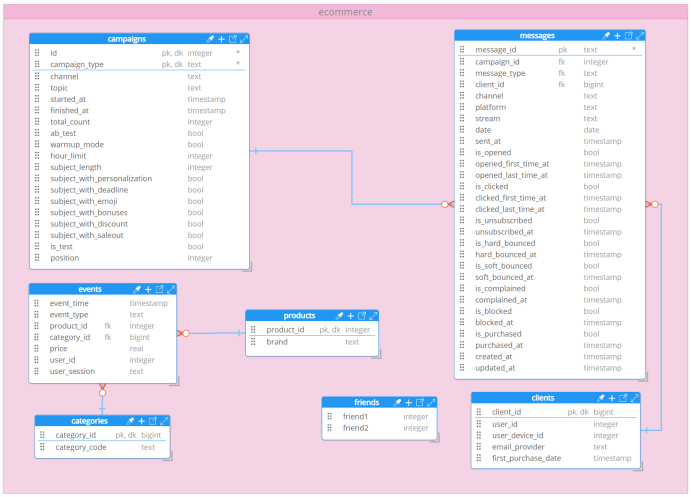

Fig. 1. PostgreSQL model

- In `campaigns` table we had to convert float values to int once for columns `total_count`, `hour_limit`, `subject_length` and `position`.
- Also in `campaigns` table feature `ab_test` contained only values `null` or `True`, thus, we had to convert this nulls to False.
- In `messages` table columns `is_opened`, `is_clicked`, `is_unsubscribed`, `is_hard_bounced`, `is_soft_bounced`, `is_complained`, `is_blocked`, `is_purchased` contained values `t` and `f`, which had to be substituted with `True` and `False`.

In this section, we can also mention processes such as creating new tables in a way that does not lose data and removes all duplicates or substitutes null values if there is an opportunity.

At the end, we had 7 tables: `campaigns`, `messages`, `clients`, `friends`, `events`, `products` and `categories` to work with. All data models would have similar structure based on this tables.

C. PostgreSQL

Fig. 1 contains general structure for PostgreSQL data model. To build an optimal database in postgres, it is necessary to

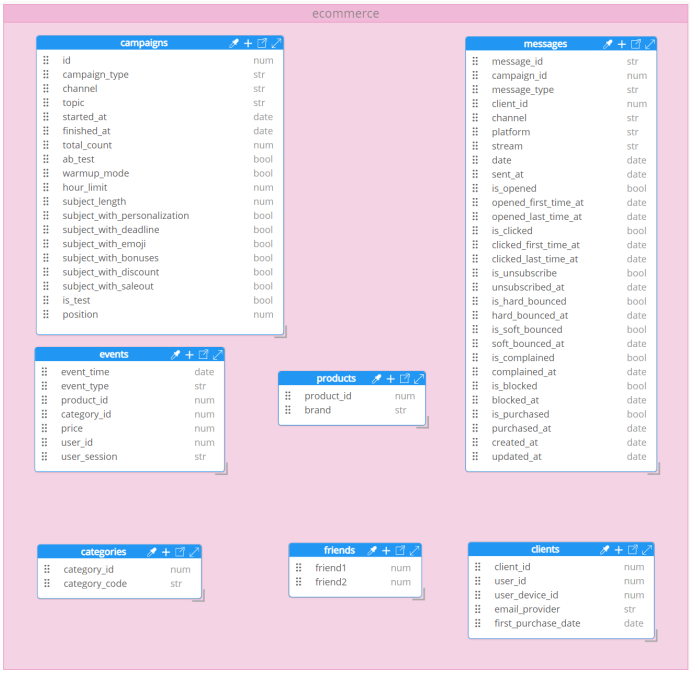

Fig. 2. MongoDB model

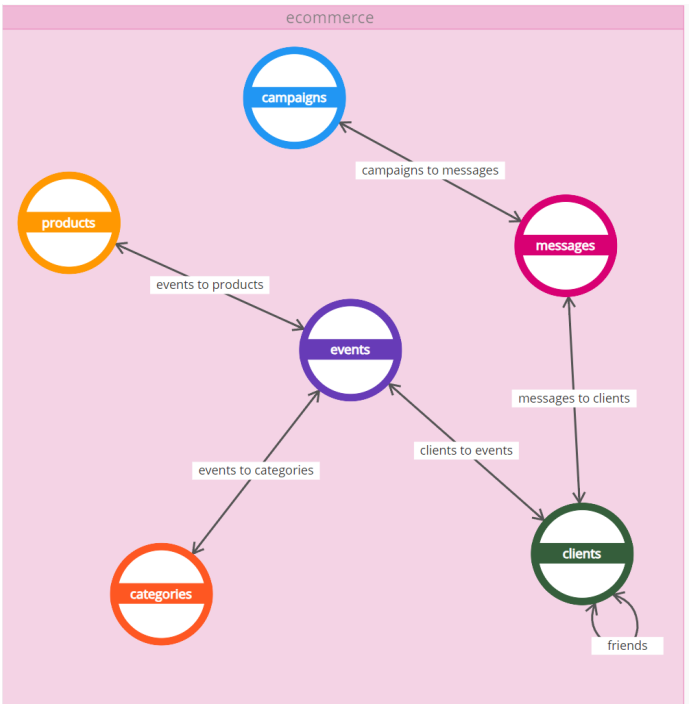

Fig. 3. Neo4j model

determine the keys and relationships between the tables. Let's start with primary keys:

- Table `campaigns` has composite key based on columns `id` and `campaign_type`.
- Table `messages` has primary key `message_id`.
- Table `clients` has primary key `client_id`.
- Table `products` has primary key `products_id`.
- Table `categories` has primary key `categories_id`.

Based on that primary keys we can define foreign keys show connections between tables:

- For table `messages` fields `campaign_id` and `message_type` are fk which are connected to table `campaigns`.
- For table `messages` field `client_id` is a fk which are connected to table `clients`.
- For table `products` field `product_id` is a fk which are connected to table `events`.
- For table `categories` field `category_id` is a fk which are connected to table `events`.

After that we can import our preprocessed data from csv files in the PostgreSQL.

D. MongoDB

Fig. 2 contains general structure for MongoDB data model. As you can see, it's almost the same as in PostgreSQL. However, we faces with several difficulties on importing stage for this noSQL database. First of all, we had to build json files, since MongoDB can't determine that the provided boolean or datetime field is not a string. Even though, we have to rebuild our jsons and manually set for all timestamp fields value in format of `$date: %Y-%m-%dT%H:%M:%SZ` and convert each of them in corresponding time format.

E. Neo4j

Fig. 3 contains general structure for Neo4j data model, which differs only in putting information from `friends` table into relationship between clients. As for connection between nodes, it's very similar to one in PostgreSQL (f.e. `messages` connected with `clients` nodes via `client_id` property and etc.), however, apart from connection between `clients` node via `user_id` field, there is also a new connection from `events` to `clients` via `user_id field`.

I want to highlight the importance of indexes for Neo4j fields which are used to build relationships. Without them importing process requires numerous amount of time.

Also, to import all the data, I had to modify neo4j.conf file to increase memory limits for executions.

II. Discussion

I successfully created data models for PostgreSQL, MongoDB and Neo4j databases and imported data in them. Unfortunately, I did not have time to compare the effectiveness of these models on different queries. It took too long to prepare the environment, models, data, and import data into the databases themselves. Due to the large amount of data in the source files, processing required significant computing resources. Moreover, one error could lead to a long wait for

new data to be prepared or uploaded. In addition, it was by no means easy to find a way to correctly import a date from various formats into MongoDB. If there was less data, at least we wouldn't have to wait so long for it to be processed. My opinion is that this assignment requires too much time and efforts for the amount points that it can give.

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.