Big Data IU-S25 Assignment 2 Report

Dmitry Beresnev
Innopolis University
Innopolis, Russia
d.beresnev@innopolis.university

***Abstract*—The task of this assignment was to design a data models for e-commerce company to efficiently manage and analyze large amount of data. During the assignment, Relational, Document and Graph data models were implemented. Also, the hybrid model was designed, which has potential to outperform single-database solutions. For hybrid model, advantages and disadvantages were discussed. Business queries for three proposed single-database designed were designed, implemented and tested. Finally, the designed query were benchmarking and results analyzed.**

I. Introduction

A global e-commerce company, faces challenges managing exponential data growth from customer transactions, product catalogs, social networks, and inventory systems. The company wants to find an efficient way to store and analyze the data. They want to see which data model they can use to have fast data retrieval and query processing.

The company is struggling with three big data issues:

- **Volume**: large amounts of data per day
- **Variety**: Social network data, user behavior events, and campaign messages
- **Value**: Analyzing customer social networks for targeted marketing

The primary objective of this assignment is to design a big data solution to store petabytes (the provided dataset is only a snapshot) of raw data using effective, optimal and scalable data models that can handle growth and complexity without compromising quality, efficiency, or usability. The optimal design will enable analytics and transactions for real-time and batch data processing. Given task is to find the best data model for storing the big data and obtaining valuable insights from it.

II. Data Modeling

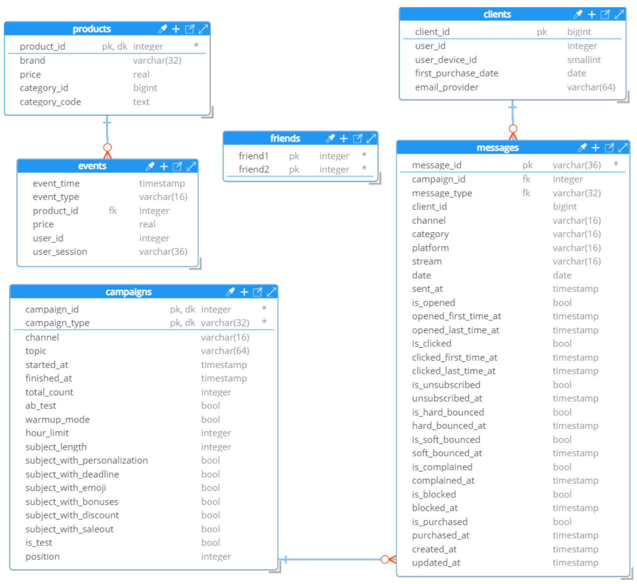
You can find the e-commerce dataset description in the assignment description from 'reference' directory. Three data models, each for specific database, were designed: Relational (for PostgreSQL), Document (for MongoDB) and Graph (for Neo4j). All the models are designed using Hackolade.

A. Relational Data Model

The first database version is modeled for PostgreSQL. There are six tables in the relational model, shown in Figure 1: 'products', 'events', 'campaigns', 'friends', 'clients' and 'messages'. This design has the following main features, some of which are inspired by notion of normalization:

- Clients are placed in separated table, what removes data redundancy in 'message' table. Here i assume that email provider is uniquely defined by client, which lines with dataset description and collected insights
- Products with related fields are separated into new table to remove data redundancy in 'events' table
- In table 'campaigns', campaign id ad campaign type are used as primary key, as how is stated in dataset description, campaign id is unique ony within one campaign type. This composite primary key produces corresponding composite foreign key in table 'messages', where message type corresponds to campaign type. Once again, this decision does not conflict the dataset description and collected insights, as both types have same set of values
- Note, that 'event' table, like 'products', also has price field. That is because event, like view or purchase, could occur when the price of product was different from the current product price. This feature is also introduced in other models

Other relations, together with data types and foreign and primary keys, shown in Figure 1 were deduced from dataset description and are quite natural, so there is no need to describe them in details.



products			
product_id	pk, dk	integer	*
brand		varchar(32)	
price		real	
category_id		bigint	
category_code		text	

events		
event_time		timestamp
event_type		varchar(16)
product_id	fk	integer
price		real
user_id		integer
user_session		varchar(36)

friends			
friend1	pk	integer	*
friend2	pk	integer	*

clients		
client_id	pk	bigint
user_id		integer
user_device_id		smallint
first_purchase_date		date
email_provider		varchar(64)

campaigns			
campaign_id	pk, dk	integer	*
campaign_type	pk, dk	varchar(32)	*
channel		varchar(16)	
topic		varchar(64)	
started_at		timestamp	
finished_at		timestamp	
total_count		integer	
ab_test		bool	
warmup_mode		bool	
hour_limit		integer	
subject_length		integer	
subject_with_personalization		bool	
subject_with_deadline		bool	
subject_with_emoji		bool	
subject_with_bonuses		bool	
subject_with_discount		bool	
subject_with_saleout		bool	
is_test		bool	
position		integer	

messages			
message_id	pk	varchar(36)	*
campaign_id	fk	integer	
message_type	fk	varchar(32)	
client_id		bigint	
channel		varchar(16)	
category		varchar(16)	
platform		varchar(16)	
stream		varchar(16)	
date		date	
sent_at		timestamp	
is_opened		bool	
opened_first_time_at		timestamp	
opened_last_time_at		timestamp	
is_clicked		bool	
clicked_first_time_at		timestamp	
clicked_last_time_at		timestamp	
is_unsubscribed		bool	
unsubscribed_at		timestamp	
is_hard_bounced		bool	
hard_bounced_at		timestamp	
is_soft_bounced		bool	
soft_bounced_at		timestamp	
is_complained		bool	
complained_at		timestamp	
is_blocked		bool	
blocked_at		timestamp	
is_purchased		bool	
purchased_at		timestamp	
created_at		timestamp	
updated_at		timestamp	

Fig. 1. Relational Tables for E-commerce

B. Document Data Model

The second database version is modeled for MongoDB. It creates a unique id by default automatically for each document whenever it is inserted into the database, so there was no need to set special ‘ObjectID’ type to custom identificators.

The main feature of document-based databases is that the data is represented in semi-structured documents: so it is possible to omit some fields if necessary and easily introduce arrays. I have decided to use these feature in the second design. Figure 2 shows five collections: ‘users’, ‘clients’, ‘messages’, ‘campaigns’, ‘products’. Main features of proposed design are the following:

- Clients and products are placed in separated tables, as in relational model for same reasons
- Document-based model allows to use nested structures, so I decided to combine several fields into sub-documents based on their meaning. For example, fields related to subject within table ‘campaigns’ were combined into field ‘subject’. Such design produce of more natural and intuitive way of data representation and subsequent querying
- Tables ‘friends’ and ‘events’ from relational model were transformed into collection ‘users’, as both tables refers to user id and describe the data related to the specific user: namely his/her friends and events occurred. So the element of collection ‘users’ stores the list of friends and events for specific user

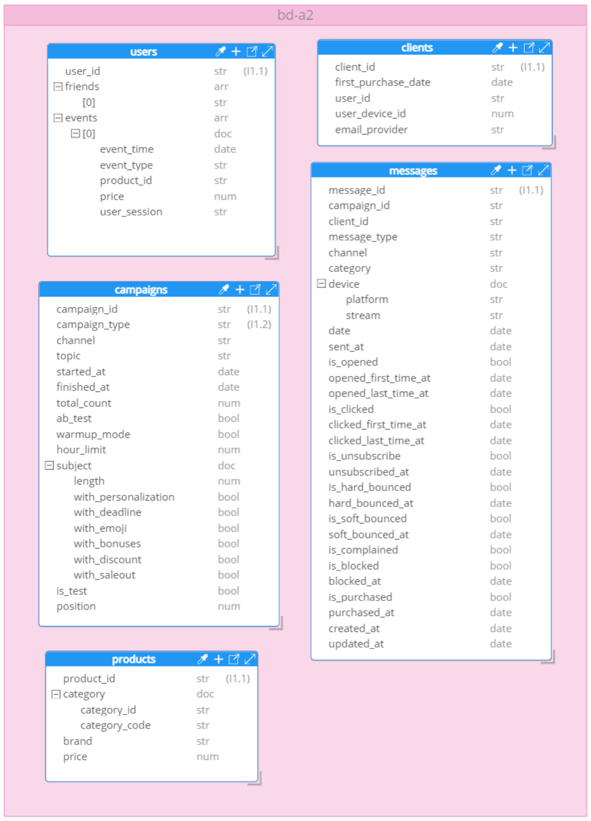
Other data types and design elements shown in Figure 2 were deduced from dataset description and are quite natural, so there is no need to describe them in details.

C. Graph Data Model

The third database version is modeled for Neo4j.

In a graph model, users and products are represented with nodes, and relationships with edges. Figure 3 shows the five types of nodes: ‘Product’, ‘User’, ‘Client’, ‘Message’ and ‘Campaign’. There also are also seven tpe of relationships: ‘VIEWED’, ‘ADDED TO CART’, ‘PURCHASED’ (between user and product), ‘FRIENDS WITH’ (between users), ‘BELONGS TO’ (between client and user), ‘RECEIVED’ (between client and message) and ‘SENT FROM’ (from message to campaign). Properties of nodes and relations are listed in Figure 4. Main features of proposed design are the following:

- Clients and products are placed in separated tables, as in relational and document models for same reasons
- Table ‘events’ were transformed to three relationship types: ‘VIEWED’, ‘ADDED TO CART’, ‘PURCHASED’, and the corresponding data transferred to relation properties. The transformation is seemed very natural, as event, in the proposed sense, is indeed a relationships between user and product
- Note that the only bi-directional relationship is ‘FRIENDS WITH’, as design of dataset does not imply one-directional friendship. Other relations has naturall hierarchical meaning



users		
user_id	str	(I1.1)
⊟ friends	arr	
[0]	str	
⊟ events	arr	
⊟ [0]	doc	
event_time	date	
event_type	str	
product_id	str	
price	num	
user_session	str	

clients		
client_id	str	(I1.1)
first_purchase_date	date	
user_id	str	
user_device_id	num	
email_provider	str	

messages		
message_id	str	(I1.1)
campaign_id	str	
client_id	str	
message_type	str	
channel	str	
category	str	
⊟ device	doc	
platform	str	
stream	str	
date	date	
sent_at	date	
is_opened	bool	
opened_first_time_at	date	
opened_last_time_at	date	
is_clicked	bool	
clicked_first_time_at	date	
clicked_last_time_at	date	
is_unsubscribe	bool	
unsubscribed_at	date	
is_hard_bounced	bool	
hard_bounced_at	date	
is_soft_bounced	bool	
soft_bounced_at	date	
is_complained	bool	
is_blocked	bool	
blocked_at	date	
is_purchased	bool	
purchased_at	date	
created_at	date	
updated_at	date	

campaigns		
campaign_id	str	(I1.1)
campaign_type	str	(I1.2)
channel	str	
topic	str	
started_at	date	
finished_at	date	
total_count	num	
ab_test	bool	
warmup_mode	bool	
hour_limit	num	
⊟ subject	doc	
length	num	
with_personalization	bool	
with_deadline	bool	
with_emoji	bool	
with_bonuses	bool	
with_discount	bool	
with_saleout	bool	
is_test	bool	
position	num	

products		
product_id	str	(I1.1)
⊟ category	doc	
category_id	str	
category_code	str	
brand	str	
price	num	

Fig. 2. Document Model for E-commerce

Other data types and design elements shown in Figures 3 and 4 were deduced from dataset description and are quite natural, so there is no need to describe them in details.

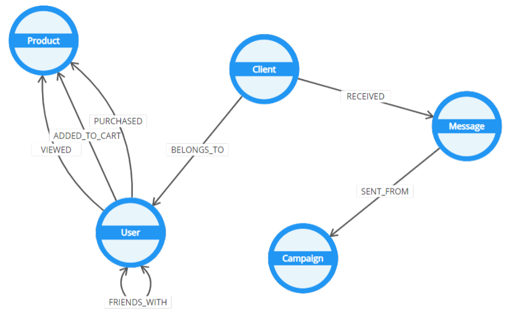


Fig. 3. Graph Nodes and Relations for E-commerce

D. Hybrid Data Model

The forth approach is to build a hybrid model, which incorporates the advantages of Relational, Document ang Graph models. Proposed hybrid model shown in Figures 5 to 8 can potentially be better than all models described above. The main characteristics of this approach are the following:

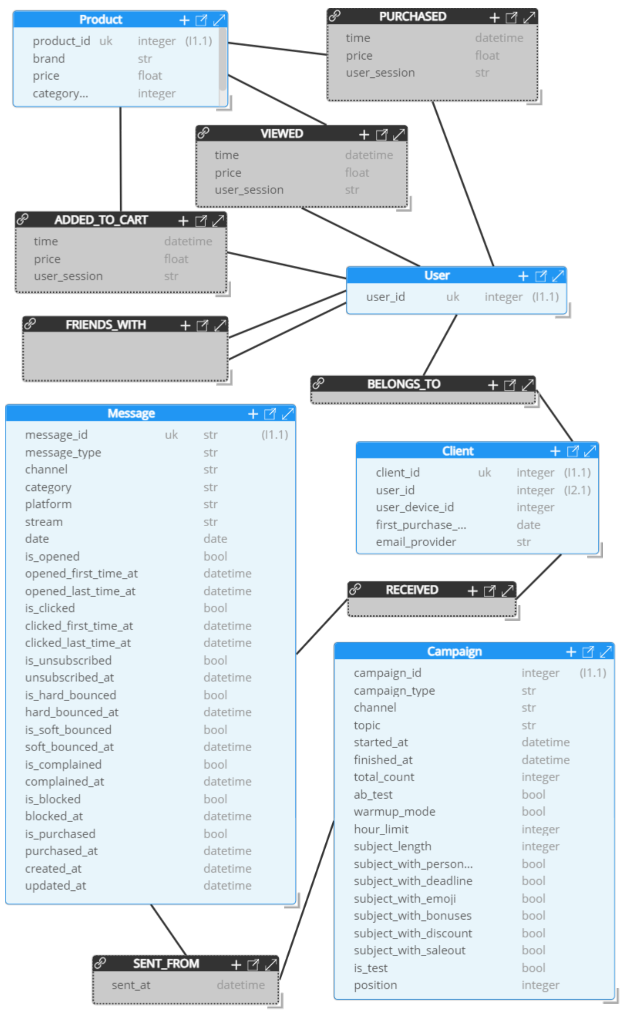


Fig. 4. Graph Model for E-commerce

- Relational part handles the structured and transactional information, together with transactional relationships, introduced by primary and foreign keys
- Document part contains flexible and event-driven data. Fields of this part can be omitted, and new fields can be added on demand. Also, nested documents are introduced for convenient grouping by meaning within items
- Graph part effectively handles relationships, the number of which can be enormous. Specifically, it implements friendship, events and user-client relationships.
- Note that 'event id' is introduced to effectively separate different information about events among Document and Graph parts. For example, Graph part can be used to determine certain event between user and product, and Document part stores information about this event, which can vary depending on event type (though in our case it is the same)

Other design features have been already described in corresponding sub-sections, and are mostly natural.

Advantages of hybrid model:

- **Optimization for different purposes**. Relational part can handle transactional data, Document part — event-driven (and testing) data, and Graph — relations-related, such as recommendations
- **Scalability and Performance Trade-Off**. Relational part preserves necessary structure, while Document and Graph parts can easily scale horizontally

Disadvantages of hybrid model:

- **Complexity**. Maintaining consistency across three databases is complex task
- **Operational overhead**. Expertise in three databases and different types of infrastructures can result in operational inefficiency
- **Slow cross-db communication**. Queries related to multiple databases will usually be slower than in case of one database

A s a result I would recommend using the hybrid model either for small project, where maintaining the consistency is relatively easy, or in quite professional and skill team, which can handle such complex structure.

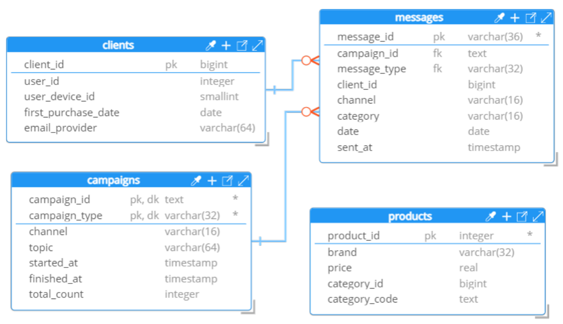


Fig. 5. Hybrid Model for E-commerce. Relational part

III. Experimental Results

A. Query Building

a) Q1: The purpose of the first query was to find out whether campaigns attracted the customers to purchase products. The business aims to engage more customers in next campaigns by leveraging the social network info. I decided to implement query for obtaining the following fraction: $f = \frac{P_p/T_p}{P_n/T_n}$, where P_p — number of messages with 'is purchased' set to true with corresponding campaign having personalization, T_p — total number of messages with corresponding campaign having personalization, P_n — number of messages with 'is purchased' set to true with corresponding campaign having no personalization and T_n — total number of messages with corresponding campaign having no personalization. If the fraction $f \geq 1$, it would mean that personalization in campaigns indeed attracted the customers to purchase products, so it is reasonable to leverage the social network info.

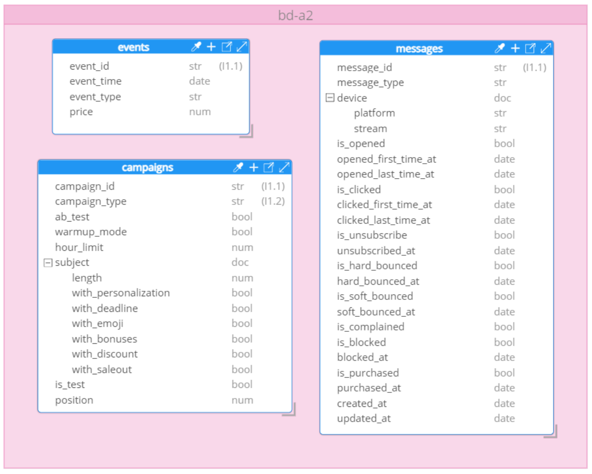


Fig. 6. Hybrid Model for E-commerce. Document part

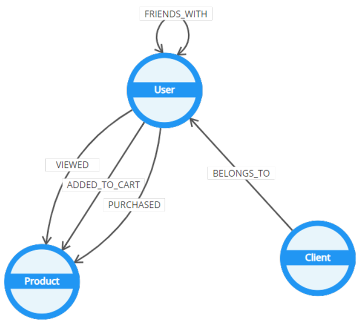


Fig. 7. Hybrid Model for E-commerce. Nodes and relations of Graph part

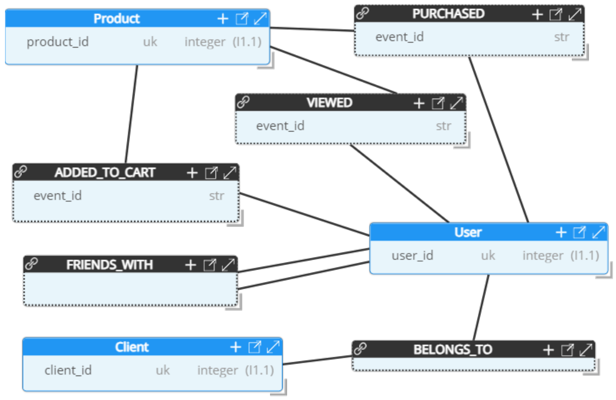


Fig. 8. Hybrid Model for E-commerce. Graph part

b) Q2: The purpose of the second query was to find top personalized recommended products we can display in the home page of the user. I decided to fix 'user id' to 563016948 for queries. To find personalized products, query firstly find the project with which the user interacted mostly. Interaction is an event of type 'view' or 'cart'. Then, for such products the overall popularity is computed as a number of events occurred with these product among all users. Finally, the results are sorted firstly by number of interactions, the by popularity and the highest 10 is selected.

B. Query Results

Queries are stored in the *scripts/queries* folder. All the queries return the same result as demonstrated in *scripts/run_queries.py*.

a) Q1: For the fist query, $f \approx 2.175$, what means personalization in campaigns indeed attracts the customers to purchase products.

b) Q2: The top products has the following identifications: [1005135, 28717064, 7004807, 1003317, 7005751, 1005105, 1004258, 1004839, 1005112, 7004492]. Most of the popular products has category code 'electronics.smartphone'.

C. Benchmarking

The queries were benchmarked using hyperfine with three warm-up runs and total number of runs equals five. System specifications were the following:

- **OS**: Window 11 10.0.26100
- **RAM**: 16 Gb
- **CPU**: Intel(R) Core(TM) i7-8750H @ 2.25GHz
- **PSQL version**: 16.0
- **Neo4j version**: 2025.02.0
- **MongoDB version**: 6.0.5
- **Mongosh version**: 1.8.0

The results of benchmarking are shown in Figure 9.

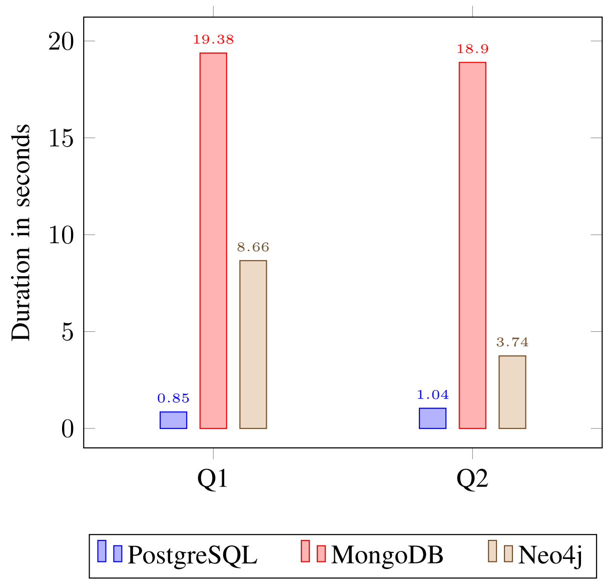


Fig. 9. Query execution results on the average of five runs

Superiority of PostgreSQL is obvious. I suppose that slow work of MongoDB is due to large amount of data, as Document models usually works worse n such cases in comparison with Relational Models. Also, the problem can be with proposed models.

Anyway, the PostgreSQL is the obvious leader, so I suggest PostgreSQL Relational model as a universal choice for the current setting.

IV. Discussion

As a result of assignment the goal was partially achieved. Three data models, Relational, Document and Graph, were designed, implemented and discussed. Also, the hybrid model was successfully designed, although, in my opinion, it is not quite applicable in the current settings, what was discussed in corresponding section. Only two out of three business queries were designed, implemented and tested. The main conclusions of query testing is that for the proposed setting, the Relational data model with PostgreSQL is more suitable, effective and universal. Finally, benchmarking was applied, and the leader, Relational PostgreSQL Data model, was determined.

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.