Relational Database and NoSQL Inspections using MongoDB and Neo4j on a Big Data Application

Conference Paper · September 2022

DOI: 10.1109/UBMK55850.2022.9919589

1 author:

Serhat Uzunbayir
İzmir University of Economics
12 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Relational Database and NoSQL Inspections using MongoDB and Neo4j on a Big Data Application

Serhat Uzunbayir
Department of Software Engineering
Izmir University of Economics
Izmir, Türkiye
uzunbayir.serhat@ieu.edu.tr

2022 7th International Conference on Computer Science and Engineering (UBMK) | 978-1-6654-7010-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/UBMK55850.2022.9919589

***Abstract*—The importance of big data, one of the most popular and researched topics of today, has been a subject of constant debate. Big data appear in almost every aspect of our lives, such as healthcare, education, shopping, social media, and industry; however, its storage and processing is not very efficient when using traditional methods. Therefore, the aim of this study is to obtain big data using a novel social shopping application that collects data from its users. The application is designed to collect information about people, products, and friendships among people, as well as product relationships, and post creations, and also enables setting up various options to mark products, such as like, buy, have, or help. The data gathered by the system is analyzed using both relational and non-relational databases, and the performance of these databases is then compared using specific queries, to reveal the model which performs better and more efficiently. Three different database models were designed and implemented for the system; a relational database, a document database, and a graph database. As a result of the experiments, there is no single model that is superior to the others as all three databases have their advantages and disadvantages. Therefore, the database selection should be decided based on the application domain.**

***Keywords*—big data, relational databases, nosql, document database, graph database**

I. Introduction

Big data is one of the most popular concepts of the last decade after being introduced in the field of astronomy and genetics [1]. The discovery of the internet dramatically changed the way data is created and processed. The sources of data come from many different environments, such as research, healthcare, the internet, social media, sensors, etc. [2]. An enormous data stack that is ready to be analyzed is produced by constant interactions in social media accounts, search engines, bank account transactions, blogs, e-mails, various types of sensors, and the interactions of users with the internet using mobile devices. To convert data into big data, it should be analyzed, classified, and transformed into a meaningful form.

There is no doubt that relational database management systems are powerful when it comes to managing small or medium-sized data. Considering the increased amount of data today, storing, analyzing, and managing data causes struggles with relational models. Database experts also point out that storing and processing big data has limitations when using traditional database management systems such as relational models [3]. The enormous variety, volume, and velocity of data makes it very difficult to store and manage big data in relational databases, and these limitations have led researchers to seek alternative approaches in order to cope with these issues. Hadoop, MapReduce, and NoSQL are current trends for big data analysis [4]. Furthermore, there are different types of NoSQL models, i.e. key-value pairs, column-family stores, document databases, or graph databases. These challenging methodologies have become a necessity for applications that use large data sets over time.

TrendPin is an online shopping application with social media features aimed at reshaping the shopping experience by involving various options for products and people. Applications that have social networking properties produce greater amounts of big data with increases in the number of elements, such as users, products, and time spent in this case. To that extent, TrendPin is modeled with three different versions: to store and manage the generated data, the first version uses a relational database, the second version, a document database, and the third, a graph database. In this study, we present these three models and analyze their performance. Please note that TrendPin is not currently active, and therefore, previously collected data from its alpha version and randomly generated additional data is used as a big data source for the experiments of this research.

The main objective of this paper is to summarize NoSQL models, review current related work, and then propose a relational, a document, and a graph model for TrendPin. The aim is to present these models and evaluate their performance by executing several queries to produce same results for all models. At the end of the experiments, the findings are compared and analyzed to assess which model performs better on which type of queries.

The rest of this paper is organized as follows. Section 2 briefly explains relational database models, NoSQL types, and presents studies related to comparing database models using big data projects. Section 3 introduces a social shopping application TrendPin as the big data source used for the experiments. Section 4 presents three different data model

978-1-6654-7010-0/22/$31.00 ©2022 IEEE

designs for the application. Section 5 discusses experimental results and performance comparisons, and finally, Section 6 concludes the paper.

II. Related Work

In this section, the preliminary terms about relational databases, NoSQL, and four types of NoSQL are explained followed by a literature review on different database comparison studies involving big data as a source.

A. Relational Databases

A relational database stores and organizes its elements as a table with rows and columns using previously defined relationships when creating the table initially [5]. A relational model includes structured data due to fixed fitting fields for each piece of information. There are two basic components of a relational schema; a relation and a set of attributes. For example:

- *Employee(SSNo, EmployeeNo, Name, Surname, Salary)*
- *Order(OrderNo, EmployeeNo, Date, Price, Item)*

In these examples, Employee and Order are relations, the rest are attributes. A primary key defines each row uniquely in relational tables. SSNo and OrderNo are the primary keys to the previous example. To merge two tables and identify information contained in both, foreign keys are used. EmployeeNo is a foreign key since it is in both tables and can be with join operations to retrieve required data.

Relational databases have existed for more than fifty years and are still considered as efficient ways to collect, store, and manage data using Structured Query Language (SQL). However, when big data became popular, relational database models seemed to have the following challenges: the need for many join statements when retrieving data, excessive table sizes, response time reduction for fast scaling and increasingly growing data, and the ability to store unstructured data. To adapt to this situation, practitioners needed to develop new models, new approaches, and even more efficient methods.

B. NoSQL: Not Only SQL

Instead of storing data as a table and tabular relations, a NoSQL database manages data using a model. Originally, NoSQL referred to non-relational, to indicate that unstructured data, which has no predefined scheme, can be easily managed through this type of database [6]. NoSQL databases are referred to "Not only Structured Query Language" to underline that these support SQL-like querying languages. The most important feature of NoSQL systems is that they are distributed, support unstructured data, and can scale easily at any time. They are used by leading companies such as Google, Netflix, Disney+, Amazon, Twitter, and Facebook to help overcome the challenges of fast-growing data and accommodate its needs.

There are four types of NoSQL database models:

1) Key-Value Stores

A key-value store is the most basic NoSQL model and it is based on key-value pairs. In this model, there is no schema, but rather, a string, such as a name, hash, or URI to represent keys, and any type of data such as video file, image, or document to represent values. This model is based on Amazon's distributed database Dynamo. The size of key-value stores can scale and increase whenever the information increases as a hash table. The queries to retrieve and store data can be executed on keys instead of values. The most popular key-value stores as of 2022 are Amazon DynamoDB, Redis, ScyllaDB, Azure Table Storage, ArangoDB, and Aerospike.

2) Column-Family Stores

Column-family stores are based on Google's fully scalable distributed storage system called Bigtable. Instead of storing data as rows, in this model, related data is kept as columns and rows have their associated keys. Column families are represented as groups and can be retrieved together when requested. Therefore, a row in this model includes a group of families as columns. The most popular column-family stores today are Google Cloud Bigtable, Apache HBase, Cassandra, DataStax, Accumulo, and Amazon Key Spaces.

3) Document Databases

IBM Lotus Notes is the first document database that paved the way for further development of other related products. A document database can manage semi-structured data which is structured but not organized like a relational model. Data is kept as different types of documents, such as PDF, XML, YAML, or JSON in these databases, and a client can access and manage these documents at any time. The structure of documents is based on key-value pairs. However, queries can be written for both keys and values in this model. The most popular document database products as of 2022 are Some MongoDB, IBM Cloudant, Apache CouchDB, MarkLogic, Google Cloud Firestore, CrateDB, and OrientDB.

4) Graph Databases

Graph databases are based on graph theory involving nodes and edges between them. Nodes are used to designate entities, and edges are used to designate relationships. This type of database became common with the popularity of Neo4j. The most important feature of a graph database is that it can be optimized for traversals on the graph and quickly find matching patterns whenever they exist. This allows them to be optimized with social networks and social media featured domains. Furthermore, graph databases provide index-free adjacency which confirms that each node has information about its neighboring nodes, and support for semi-structured data [7]. There is no need to perform join operations, therefore the model scales as the data increases. One other advantage of these databases is that they can be easily modeled on a whiteboard for brainstorming ideas and then can be converted into graphs using related products. The power of a graph database comes from the ease of modification by adding or removing nodes or edges. To illustrate how a graph database is modeled, consider the following scenario and see its formal graph model in Fig. 1;

"Steve is friends with both Natasha and Tony. Natasha owns a Harley-Davidson LiveWire, and Tony wears Christian Dior Tie 173 Eyeglasses".

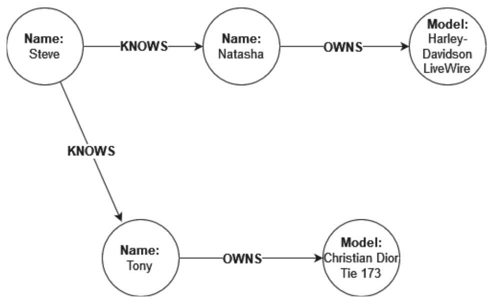

Name: Steve
KNOWS
Name: Natasha
OWNS
Model: Harley-Davidson LiveWire
KNOWS
Name: Tony
OWNS
Model: Christian Dior Tie 173

Fig. 1. Formal graph model from Fig. 2

The most popular graph database products as of 2022 are Neo4j, AllegroGraph, InfiniteGraph, Titan, FlockDB, BlazeGraph, and Apache Giraph.

For this study, the following popular database types and query languages were selected to design and experiment with:

- Relational database: Microsoft SQL Server and SQL
- Document database: MongoDB and NodeJS
- Graph database: Neo4j and Cypher

C. Literature Review

There are a number of database comparison studies in the literature, both in the format of conference papers and journals. Some propose a comparison among NoSQL types, others provide a comparison between relational and non-relational databases. We discuss selected recent studies in this section. The investigated studies were found by scanning Google Scholar, IEEE Xplore, ScienceDirect databases together with Google searches.

Jose and Abraham [8] compared and contrasted execution times of MySQL and MongoDB, showing that the performance of MongoDB is superior to that of MySQL. The result of the study suggests that the use of non-relational databases is still a current trend for big data projects. This study has similarities with our research, however, the comparison involves only two database models.

Gyorödi et al. [9] conducted a comparative experiment using CouchDB and MySQL to assess the performance of CRUD operations. They modeled two databases and measured the execution time of queries. There were two data structures for CouchDB; the first involved documents with references to each other, and the second contained all data for each entity created. In the end, the results showed that the second version performed better. On some tests, MySQL was more efficient especially considering small element retrievals due to the indexing.

Kotiranta et al. [10] analyzed database performances of Neo4j for the NoSQL database product, MySQL and MariaDB for relational database products using different query types. They used both simple and complex queries to find the most efficient approach. They found that Neo4j performed better using simple queries, but that relational database approaches performed significantly better on complex queries even though the methods are older than NoSQL approaches. Additionally, they stated that MariaDB has different feature sets, and was 90 times faster than MySQL. Our aim is similar, but we also intend to compare NoSQL models with each other.

Kanchan et al. [11] empirically evaluated MySQL, MongoDB, Redis, and Cassandra to identify the key factors to consider when choosing which database product to use with certain projects. They applied various sized datasets, and performed insert, delete, read, and update queries to measure time and the use of memory. As a result, each database had particular strengths and weaknesses according to the types of database operations.

Khasawneh et al. [12] surveyed finding alternatives to relational databases when considering big data that is increasing by volume. They aimed to compare NoSQL categories with consistency, availability, partition features, and BASE properties as opposed to relational models' ACID properties. This study provides a useful overview of NoSQL types, but lacks practical implementation to present performance-related comparisons.

III. TrendPin as a Big Data Source

TrendPin is a platform founded by Murat Demir and Kerem Işık in 2012 that completely reshapes the online shopping experience by involving social networking features. The goal of the system is to diminish hesitation when considering purchasing a product, and help the customer decide whether it is worth buying, good, or better than similar products. Users can see if any friend have the considered product and can request information about their experiences with it.

The system requires various quality of life improvements to support current trends. Therefore, it is not active currently. Its previously collected data from the alpha version, together with randomly generated additional data is used as a big data source for this study. The system allows users to connect with their friends, check what products they own, ask about the products if they already have them, personalize their profile pages, search for products to buy, or search for information on product pages, and be able to redirect to their website with a single button. Fig. 2 shows a sample product page.

There are marked parts on this figure. Part a is a picture of the product, part b is the name of the product, part c shows three buttons specific to TrendPin to mark the product as **BUY**, **LIKE**, and **HAVE**, part d is the description of the product, part e are four special buttons users can select when creating product related posts, such as *I want to buy this*, *I have this*, *I need help*, or *I like it*. Part f displays prices from various sellers, part g shows which friend has this product, and part h shows who likes this product. TrendPin is

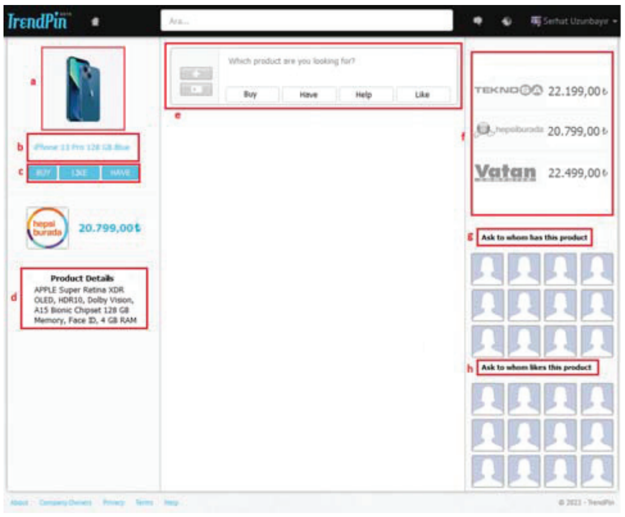

TrendPin
Ara...
Serhat Uzunbayır
Which product are you looking for?
Buy
Have
Help
Like
TEKNOSA 22.199,00₺
hepsiburada 20.799,00₺
Vatan 22.499,00₺
Ask to whom has this product
Ask to whom likes this product
iPhone 13 Pro 128 GB Blue
BUY
LIKE
HAVE
hepsiburada 20.799,00₺
Product Details
APPLE Super Retina XDR OLED, HDR10, Dolby Vision, A15 Bionic Chipset 128 GB Memory, Face ID, 4 GB RAM
a
b
c
d
e
f
g
h
About
Company Owners
Privacy
Terms
Help
© 2021 - TrendPin

Fig. 2. A sample product page of TrendPin

developed using the ASP.NET framework. More details can be found in our previous work [13].

IV. Alternative Database Approaches for TrendPin

In this section, three versions of database models for TrendPin are presented. First, the relational data model for the system is introduced; then, the document data model is discussed; and finally, the graph model of the system is presented.

A. Relational Data Model

The first database version of TrendPin is modeled using MSSQL. There are 14 tables in the relational model, shown in Fig. 3: FbFriendship, User, Product, Post, UserProductRelation, PostLike, ProductCategory, CommentLike, RelationType, Comment, FbProfile, Shop, Vendor, and Follower. This design involves some precautions to avoid normalization-related issues; querying to retrieve data sometimes requires nested joins and, as a result, response time may be increased.

B. Document Data Model

The second version of TrendPin is modeled using MongoDB. It creates a unique id by default automatically for each document whenever it is inserted into the database. Users can pre-select the type of unique id such as number. If no type is specified, an object id is assigned by MongoDB.

As previously mentioned, in document databases, data is kept as semi-structured documents. Users of TrendPin are registered through Facebook, therefore this information should also be kept through the database. Document design should be sufficiently comprehensive to support all necessary information fields and keep all information in a single document as far as possible. This enables retrieving fewer files when requested by users, and decreases the response time, since it does not necessarily return many documents. For this reason, document

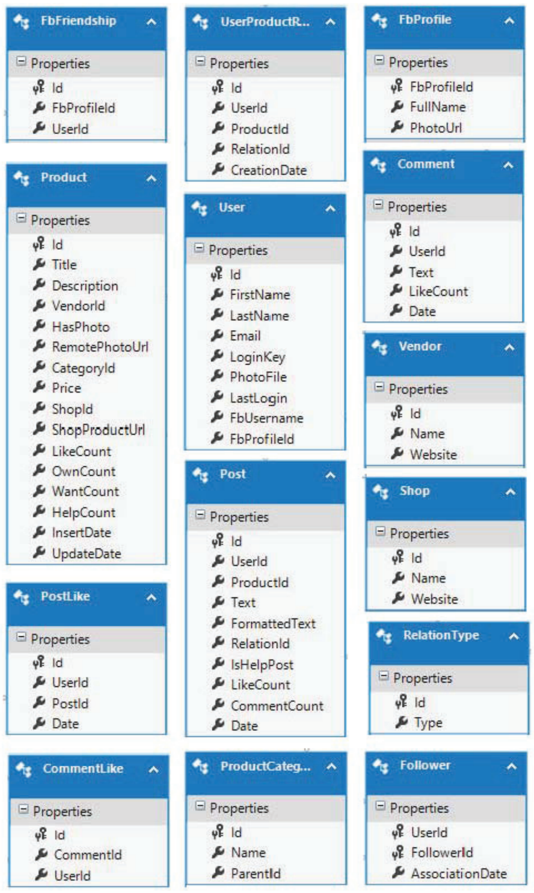
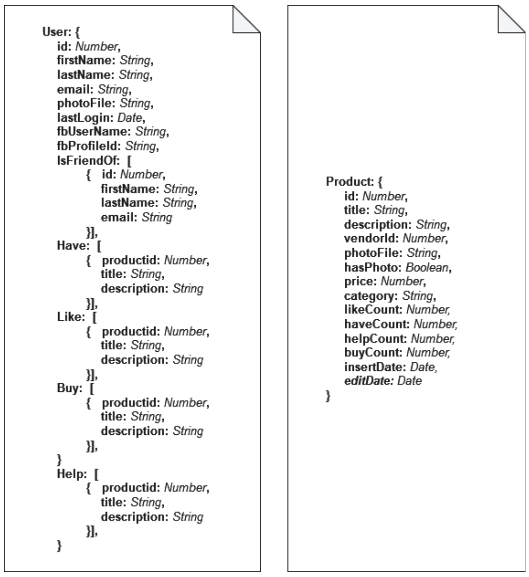


Fig. 3. Relational tables for TrendPin

design should be done carefully to increase the efficiency of database operations. Fig. 4 below shows the document model structure for user and product.


```
User: {
   id: Number,
   firstName: String,
   lastName: String,
   email: String,
   photoFile: String,
   lastLogin: Date,
   fbUserName: String,
   fbProfileId: String,
   IsFriendOf:  [
         {   id: Number,
             firstName: String,
             lastName: String,
             email: String
         }],
   Have:  [
         {   productid: Number,
             title: String,
             description: String
         }],
   Like:  [
         {   productid: Number,
             title: String,
             description: String
         }],
   Buy:  [
         {   productid: Number,
             title: String,
             description: String
         }],
   }
   Help:  [
         {   productid: Number,
             title: String,
             description: String
         }],
   }
```

```
Product: {
    id: Number,
    title: String,
    description: String,
    vendorId: Number,
    photoFile: String,
    hasPhoto: Boolean,
    price: Number,
    category: String,
    likeCount: Number,
    haveCount: Number,
    helpCount: Number,
    buyCount: Number,
    insertDate: Date,
    editDate: Date
}
```

Fig. 4. Document models in document database

C. Graph Data Model

The third database version of TrendPin is modeled using Neo4j. This model is first designed on a whiteboard while brainstorming ideas using related scenarios.

In a graph model, users and products are represented with nodes, and relationships, with edges. The overall model includes two main scenarios; relationships and friendships among users and products, and creating posts and comments. A more detailed version of this model can be found in [13].

Users and products have their key-value features much like the attributes in a relational model (See Fig. 5). Attributes are stored in nodes. Users are able to become friends with others using IS_FRIEND_OF relationship. Although the figure shows a one-way relationship, it is considered both ways. All users know their friends. Users can be related with products using **LIKE**, **HAVE**, **BUYING**, or **NEEDS_HELP** relationships.

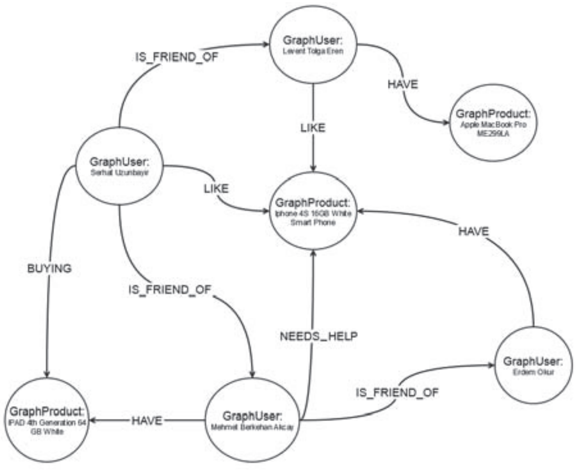


Fig. 5. Relationships and friendships among users and products

Creating posts about a product requires a selection of four options: buy, like, have, or help. The user can comment and select one of these options to create a post. After a post is created, it is added as a node called GraphPost and kept at the top of all posts with selected features (See Fig. 6).

V. Experimental Results

We previously tested relational and graph models in our own study [13]. In this study, we implemented a document database in our system and performed experiments once more. We used another PC setup, therefore the exact values on the results may be different from our previous work. Three versions of databases are compared by executing SQL, Node.JS, and Cypher queries that return the same results. The execution time of queries is calculated to reveal how a particular type of database performs according to the type of operations. We performed experiments using various CRUD operations: five reads, an insert, two updates, and a delete operation.

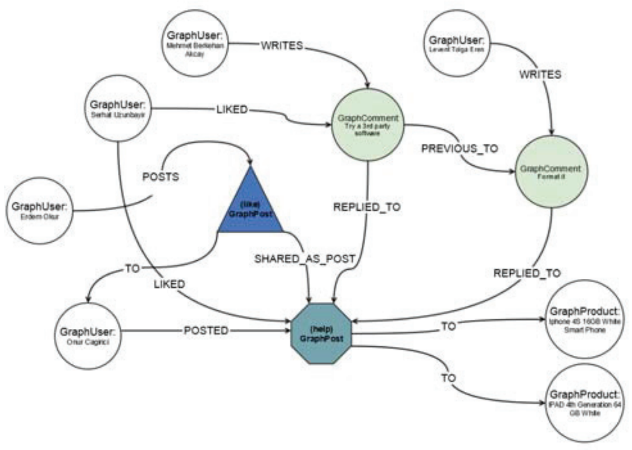


Fig. 6. Creating posts and comments

Each query had been executed 500 times on a PC that had Windows 10 Enterprise 64-bit OS, 16 GB ram, Intel Core i7-8700k CPU at 3.70 GHz processor. For relational, document, and graph models, we used Microsoft SQL Server 2019, MongoDB 5.0.9, Neo4j 4.3 respectively. The average test results are illustrated together in Fig. 7.

- **TEST 1:** This test aims to measure how a UNION from SQL query performs against others. We want to *"find the list of member friends of users given by a user id"*.
- **TEST 2:** This test aims to access results from a single type of matching. Multiple joins used for SQL for this test. It is aimed to *"find the first five products a user has ever selected as **BUYING**"*.
- **TEST 3:** This test aims to find results from multiple types of matching with sub queries. Join operations for relational model and two times matching for others. Here, we want to *"find the shops of the last two products which a user selected as **NEEDS_HELP**"*.
- **TEST 4:** This test aims to retrieve various types of data at once. A post can be a text or other types. We want to *"find all posts of a user given by a user id"*.
- **TEST 5:** This test aims to test traversals using friendship information. We want to *"find last ten users who became friends with the user given by a user id"*.
- **TEST 6:** This test aims to insert a new user with a condition. We want to *"insert a user with a friendship of at least three other users, and buying a car"*.
- **TEST 7:** This test aims to update a single product attribute. We want to *"update the price of a product from a specific vendor and the name of the product ends with 'NE'"*.
- **TEST 8:** This test aims to update many users at the same time. We want to *"update three users' friendship status"*.
- **TEST 9:** This test aims to delete an entity completely. We want to *"delete a product given by a user who has it and its price is greater than 5000 TL"*.

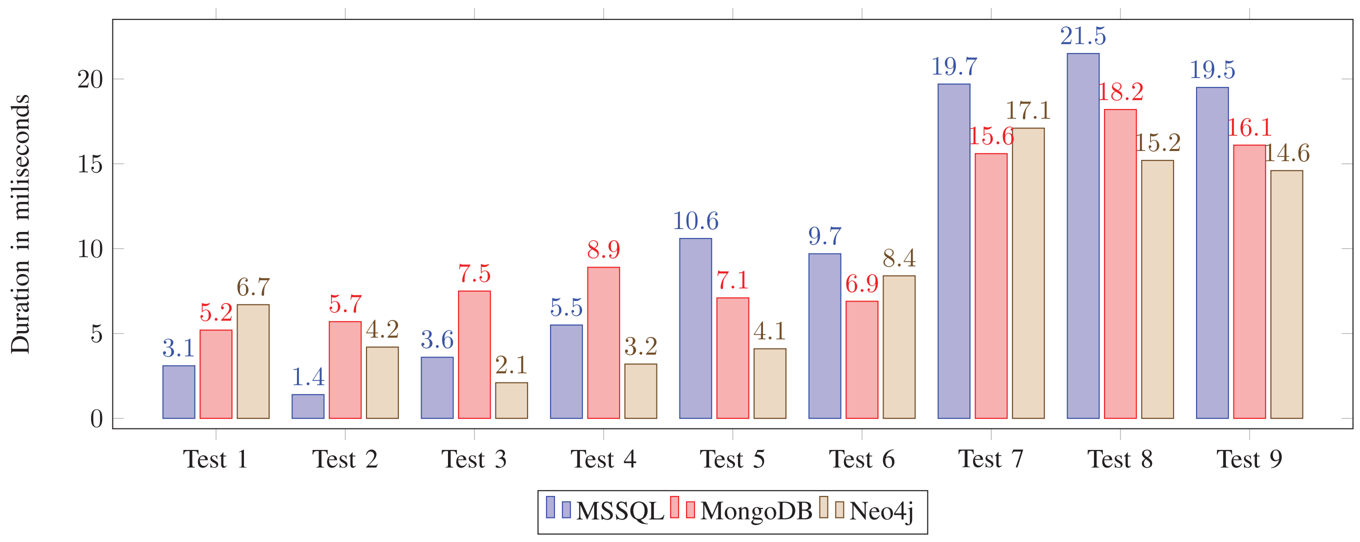


Fig. 7. Query execution results on the average

The results indicate there is no database model to achieve all nine tasks faster than the others. MSSQL performs the best when the task is to merge information scattered around different entities and match a single type of data. Neo4j works faster for queries such as multiple types of matches, traversals to get the first or the last couple of records, updating many records at once, and deleting entities. MongoDB performs more efficient when inserting, and updating single entities.

VI. Conclusions

In this study, we analyzed and compared three database models using social shopping application. Before the analysis, we reviewed relational databases and NoSQL trends. We discussed a big data source, TrendPin, and proposed three different database models for it: a relational model, a document model, and a graph model. We also discussed the implementation details for these technologies. Finally, we performed and discussed nine experiments on these databases using queries of SQL, Node.JS, and Cypher.

In our previous work [13], we showed that neither the relational nor the graph model is superior to the other. In this study, we implemented a document model on the same project, and tests revealed that overall, there is no database that is consistently better than others. In some tests, MSSQL performed better, whereas, in others, Neo4j or MongoDB performed more efficient. Our claim still persists: important factors affecting database performance are; the domain of the application, and data model design choices, and query types. For a future work, other database types or products, such as HBase can be tested with the same queries of this research.

References

[1] E. D. Feigelson and G. J. Babu, "Big data in astronomy," *Significance*, vol. 9, no. 4, pp. 22–25, 2012.

[2] D. Blazquez and J. Domenech, "Big data sources and methods for social and economic analyses," *Technological Forecasting and Social Change*, vol. 130, pp. 99–113, 2018.

[3] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, "Sql versus nosql movement with big data analytics," *International Journal of Information Technology and Computer Science*, vol. 8, no. 12, 2016.

[4] R. Kumar, B. B. Parashar, S. Gupta, Y. Sharma, and N. Gupta, "Apache hadoop, nosql and newsql solutions of big data," *International Journal of Advance Foundation and Research in Science & Engineering (IJAFRSE)*, vol. 1, no. 6, pp. 28–36, 2014.

[5] A. DeBarros, *Practical SQL: A Beginner's Guide to Storytelling with Data*. No Starch Press, 2022.

[6] R. P. Padhy, M. R. Patra, and S. C. Satapathy, "Rdbms to nosql: reviewing some next-generation non-relational database's," *International Journal of Advanced Engineering Science and Technologies*, vol. 11, no. 1, pp. 15–30, 2011.

[7] R.-G. Urma and A. Mycroft, "Source-code queries with graph databases—with application to programming language usage and evolution," *Science of Computer Programming*, vol. 97, pp. 127–134, 2015.

[8] B. Jose and S. Abraham, "Performance analysis of nosql and relational databases with mongodb and mysql," *Materials today: PROCEEDINGS*, vol. 24, pp. 2036–2043, 2020.

[9] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, R. Ş. Győrödi, G. A. Gabor, and G. D. Pecherle, "Performance analysis of nosql and relational databases with couchdb and mysql for application's data storage," *Applied Sciences*, vol. 10, no. 23, p. 8524, 2020.

[10] P. Kotiranta, M. Junkkari, and J. Nummenmaa, "Performance of graph and relational databases in complex queries," *Applied Sciences*, vol. 12, no. 13, p. 6490, 2022.

[11] S. Kanchan, P. Kaur, and P. Apoorva, "Empirical evaluation of nosql and relational database systems," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 8, pp. 2637–2650, 2021.

[12] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, "Sql, newsql, and nosql databases: A comparative survey," in *2020 11th International Conference on Information and Communication Systems (ICICS)*, pp. 013–021, IEEE, 2020.

[13] S. Uzunbayir, "A comparison between relational database models and nosql trends on big data design challenges using a social shopping application," Master's thesis, Izmir University of Economics, 2015.