Position and Communication Papers of the 16th Conference on Computer Science and Intelligence Systems pp. 133–139 DOI: 10.15439/2021F33 ISSN 2300-5963 ACSIS, Vol. 26

A Comparison between a Relational and a Graph Database in the Context of a Recommendation System

Liana Stanescu
University of Craiova, Faculty of Automation, Computers and Electronics, Craiova, Romania
Email: stanescu@software.ucv.ro

***Abstract*—This paper presents a comparison between relational and graph-based database systems' performance in a modern web application recommendation system. The comparison is conducted on five different queries starting with simple ones, leading up to more complex queries, that are performed in a typical web social application. The implementation is done in C# using .NET framework and the database systems used are SQL Server and Neo4j. For the comparative study we used a database designed in the context of a recommendation system for a culinary application. In order to effectively test the performance of both graph and relational database systems, tests were performed on four data sets that contain 350.000, 700.000, 1.400.000 and 2.100.000 entries. The tests imply performing five different retrieval queries taken in order of difficulty both in SQL and Neo4J.**

I Introduction

WEB applications and mobile applications have gained popularity in recent years, being user-friendly, offering commodity and an easy to use environment for research, reading, buying and so on.

Considering the fact that users often prefer the use of a mobile or a web application, over physical resources, for quick information search, the creation of web applications that rapidly deliver information based on user filtering seems natural and has become well spread.

However, this might not be enough for users, who are eager to rapidly learn about an item based on their preferences, without having to search for particular criteria lead the path for development of more complex recommendation systems.

For a majority of people, especially individuals living in an urban environment, time-consuming activities retain them from spending time researching. This can be avoided by the development of online web application that offers fast and innovative ideas based on users' habits. The recommendations, in the form of responses to users, need to be delivered fast, no matter how complex the application becomes, as it is a requirement implied by the fast-paced living era.

A critical aspect to consider is the database where the information will be stored. There are classical solutions like relational databases or the more recent NoSQL solutions, as graph databases. A well conducted research is mandatory when choosing the database fit for the application [1].

Different aspects such as what type of database will fit the application, what kind of structure the database will have and how fast will it be able to deliver data to the users depending on its structure are important to be taken into account.

Because in the literature we found few similar studies, we decided to experiment with the use of both a relational database and a graph type to analysis which is in this context the appropriate solution.

The paper is thus organized: section 2 presents related work, section 3 briefly introduces graph databases, section 4 contains the experiments and section 5 shows the conclusions.

II related work

In the last few years, the focus shifted from typical applications to ones that are focused on the users and their preferences. Clearly, the most used applications nowadays are social media platforms, so the attention shifted from relational databases to more appropriate database systems. There is research done is this area, since graph databases are gaining more and more ground every day and choosing the proper system has an enormous impact on the application's functionality and response time [6].

Surajit Medhi and Hemanta K. Baruah in [7] create a similar comparison between a relational and graph database performance on a simple Cricket application reaching similar results in favor of the Neo4J system. However, their tests are performed on only 400 objects and 3 queries with a decreased difficulty.

In [8] the authors present a similar comparative study in the context of a cancer treatment application. They compared the performance of a relational database implemented in MySQL and a graph database implemented in Neo4J. The comparison was made on twelve queries and three datasets: 1000, 10.000

and 100.000 records. The results indicate that MySQL performs better than Neo4J in most cases, but Neo4J is better when the queries involve multiple joins between tables and the number of records is 100.000.

In [11] the authors review the literature of recent years that have analyzed in detail the NoSQL databases and relational ones in order to highlight the characteristics of each type of database, especially for NoSQL technology that appears as a new solution over relational databases.

Another article [10] presents the results of the comparison between Oracle relational database and NoSQL graph database. The comparison was made in two directions: the first direction aimed at executing queries in the types of databases and the second direction involved performing a predictive analysis on the experimental results.

Given that both relational and graph databases can manage both relational and graph data, other researchers have tried to establish the limitations of these two technologies. In [9] they present their conclusions of the experiments that involved a unified benchmark for relational and graph databases over the same datasets using the same queries and the same metrics.

In a more recent article, the authors compared the performance of MySQL and Neo4J databases regarding the memory usage and execution time. The results highlighted the following: MySQL has a faster execution time than Neo4J, both these databases have the same time complexity, Neo4J has a higher memory usage than MySQL and Neo4J has better flexibility than MySQL [12].

This activity of comparing the relational and NoSQL databases is a current concern, as it is clear that there are applications for which relational databases are the best solution, while for other types of applications, new NoSQL technologies are preferred.

III. RELATIONAL DATABASES VS GRAPH DATABASES

Relational databases have been the basis of software applications since the 1980s, and still are [5]. Relational databases store data in a well-structured format within tables consisting of columns of certain data types and rows of those defined data types [5]. Relational databases require designers and applications to strictly structure the data used in their applications. Relational data is stored in tables, and the relationships between them are made simply through referential integrity which involves the presence of the external key that refers to a primary key [5]. To retrieve the data from several linked tables, the JOIN operation is used at query time by matching primary and foreign keys of all rows in the connected tables. These operations involve large processing capacity and memory usage, having an exponential cost [5]. If the data modeling implies the existence of many to many relationships, in the relational model will appear an additional table, a so called join table with two, or more external keys, to the initial participating tables, which further increases the cost of the JOIN operation [5].

NoSQL databases have appeared, aiming to cover certain requirements of users and applications, but many of them still did not satisfy the data links optimally. Hence the need for graph databases, that are the best choice for modeling the modern world we live in [1], [5].

In the graph data model, the relationships are as important as the data themselves. As a result, there is no need to implement the links between the entities using additional concepts, such as external keys [1], [5].

Graph databases allow the creation of models that map well to the problems to be solved. In this type of database, the data looks very similar to those in the modeled mini-world, small, normalized and keeping connections. The user can query and view the data from any point of view [1], [5].

In the graph database model, each node, either entity or attribute, has a list of link records that model the links to other nodes. These relationship records are organized by type and direction and may have additional attributes. [1], [5].

This list is used by graph databases, when running an operation equivalent to the JOIN operation in the relational model, to access the connected nodes, eliminating expensive computations. In graph databases, traversing the joins or relationships is very fast because they are not calculated at query time as they are persistent [1], [5].

Neo4J is a graph database system implemented in Java and the access to data is done with Cypher Query [3], [4]. It is an ACID-compliant transactional database with native graph storage and processing [1], [5]. The relationships are materialized at creation time, which results in no penalties for complex runtime queries.

Neo4J implements the Property Graph Model in an efficient way [3] (figure 1). The property graph model is an extension of the graphs from mathematics. The following concepts are used to model a property graph:

- Nodes that are the entities in the graph
- Labels that are used to represent the role of the node; a node can have multiple labels at the same time
- Relationships that describe directed, semantically connections between two nodes
- Properties that are key-value pairs that contain information about the node or relationship.

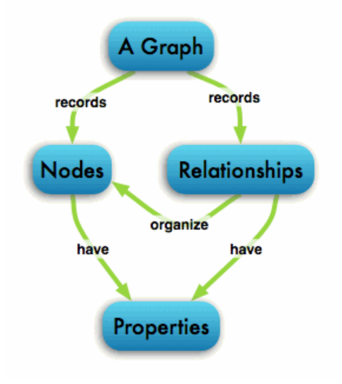


Fig. 1. Block Diagram of System Modules

The query of the relational databases is done with the help of SQL – a declarative query language. SQL commands can be used within the interfaces provided by relational database management systems, or they can be nested in an application and sent for execution to the database engine [5].

Cypher is also a declarative graph query language which is based on the basic concepts and clauses of SQL but which added a multitude of additional features specific to graphs to make it easier to work with the graph model. For describing visual patterns in graphs it uses ASCII-Art syntax. Using Cypher users can build expressive and efficient queries on graph databases [2], [5].

IV. EXPERIMENTS AND RESULTS

In order to efficiently test the performances of both graph and relational database systems, there were performed tests on four data sets as in Table I. The tests represent performing five different retrieval queries taken in order of difficulty both in SQL and Neo4J.

TABLE I. Data Sets

Set Number	Number of entries
1	350.000
2	700.000
3	1.400.000
4	2.100.000

The dataset on which the tests were performed is represented by a culinary web application and its structure can be seen in the figures below in both database systems: MS SQL Server and Neo4J.

The implementation of the application began with the development of the SQL database (figure 2) that was later exported as CSV files and imported into Neo4J (figure 3).

The database contains tables that store data about different types of ingredients, culinary recipes, and join tables that resulted from many to many relationships between data (figure 2).

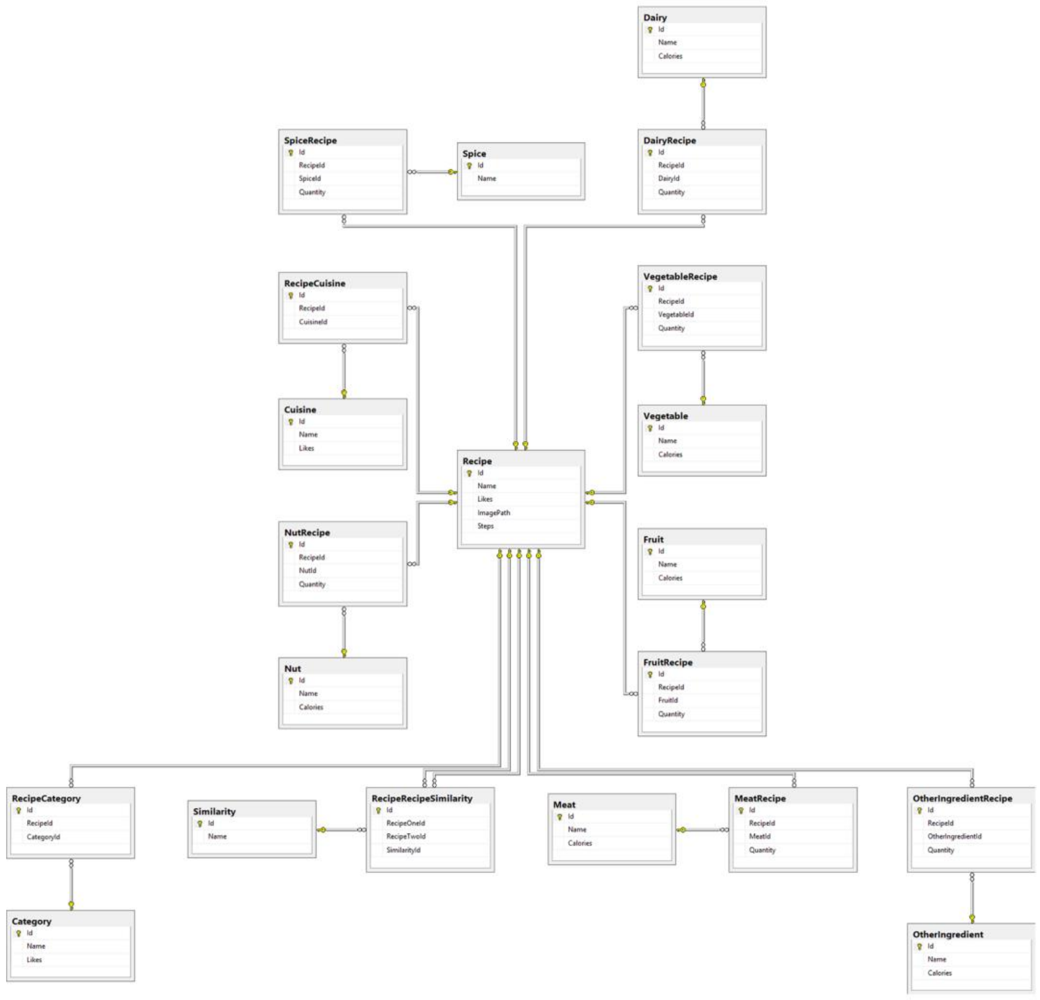


Fig. 2. Culinary App database - MS SQL Server

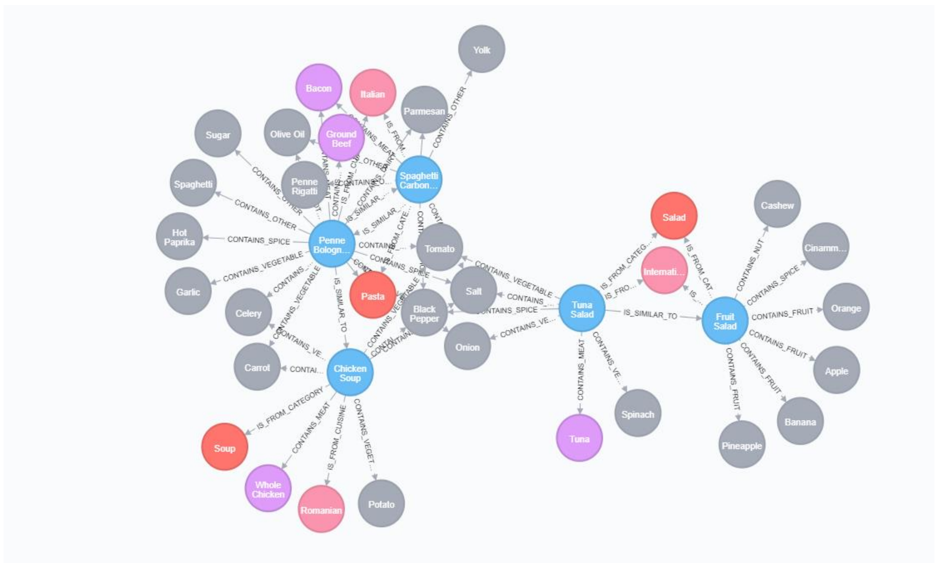


Fig. 3. A sample of Culinary App database - Neo4J

In this sample (figure 3) we can see a number of nodes and edges that represent relationships. For example the node „Chicken Soup" is related by node „Soup" with an edge called „is_from_category", by node „Romanian" with the relationship "is_from_cuisine", etc.

The tests were performed on a personal computer, in the application's solution developed in Microsoft Visual Studio 2017. The running time of the methods was measured using the Stopwatch class from the System Diagnostics namespace in the .NET Framework.

PC Configuration:

- CPU: Intel I5 @ 3.40GHz
- RAM: 8.00 GB
- OS: Windows 10 x64
- SQL Database System – SQL SERVER 2019
- Graph Database System – Neo4J Database 4.0

The connection to the SQL Database was made using Entity Framework and the connection to the Neo4J Database was possible using Neo4J Driver and Neo4J Client libraries.

Experiment 1:

Query: Get all recipes containing "Bacon"

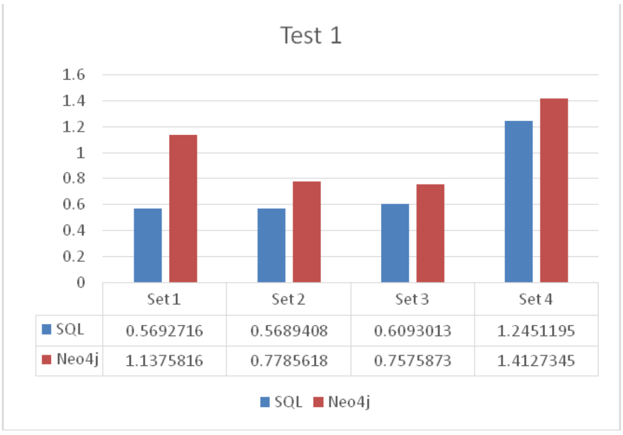
SQL Syntax

SELECT DISTINCT recipes
FROM Recipes IN Recipe TABLE
JOIN MeatRecipe IN MeatRecipe TABLE
ON RecipesId EQUALS MeatRecipe.RecipeId
WHERE MeatRecipe.MeatId EQUALS "Bacon"

Neo4j Syntax

MATCH (Recipe)- [CONTAINS MEAT]-> (Meat {"Bacon"})
return Distinct Recipe

This query involves the join of two tables and a condition. The results of the comparison appear in figure 4. It can be seen that for all four data sets, the query execution was faster in MS SQL server than in Neo4J.



	Set 1	Set 2	Set 3	Set 4
🟦 SQL	0.5692716	0.5689408	0.6093013	1.2451195
🟥 Neo4j	1.1375816	0.7785618	0.7575873	1.4127345

Fig. 4. Experiment 1 results

Experiment 2:

Query: Get all recipes containing "Bacon" from the "Italian" Cuisine

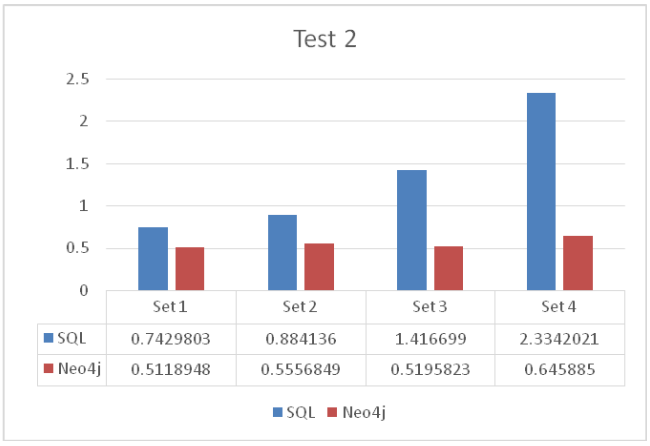
SQL Syntax

SELECT DISTINCT recipes
FROM Recipes IN Recipe TABLE
JOIN MeatRecipe IN MeatRecipe TABLE
ON RecipesId EQUALS MeatRecipe.RecipeId
WHERE MeatRecipe.MeatId EQUALS "Bacon"
JOIN RecipeCuisine IN RecipeCuisine TABLE
ON RecipesId EQUALS RecipeCuisine.RecipeId
WHERE RecipeCuisine.CuisineId EQUALS "Italian"

Neo4j Syntax

MATCH (Recipe)- [CONTAINS MEAT]-> (Meat {"Bacon"}),
(Recipe)- [IS_FROM_CUISINE]-> (Cuisine {"Italian"})
return Distinct Recipe

Query number 2 involves the join of three tables and two conditions on data. The results for this experiment appear in figure 5. In this case, in which we increased the number of joined tables, the execution time in Neo4j decreased a lot. Neo4j outperformed MS SQL server. The gap between the two systems grew larger as the number of records increased.



	Set 1	Set 2	Set 3	Set 4
■ SQL	0.7429803	0.884136	1.416699	2.3342021
■ Neo4j	0.5118948	0.5556849	0.5195823	0.645885

Fig. 5. Experiment 2 results

Experiment 3:

Query: Get all recipes containing "Bacon" from the "Italian" Cuisine from the "Pasta" Category

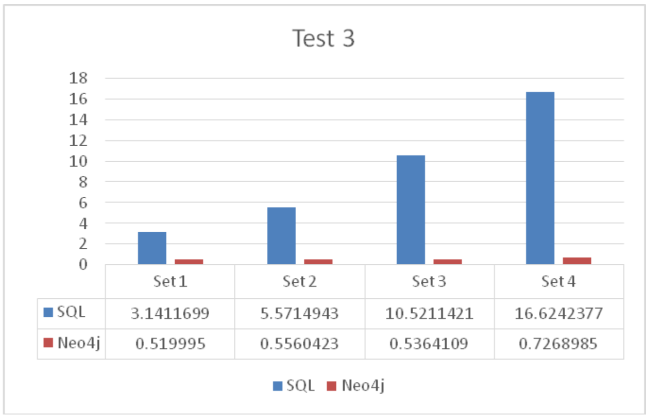
SQL Syntax

SELECT DISTINCT recipes
FROM Recipes in Recipe TABLE
JOIN MeatRecipe in MeatRecipe TABLE
ON Recipes.Id equals MeatRecipe.RecipeId
WHERE MeatRecipe.MeatId EQUALS "Bacon"
JOIN RecipeCuisine in RecipeCuisine TABLE
ON Recipes.Id equals RecipeCuisine.RecipeId
WHERE RecipeCuisine.CuisineId EQUALS "Italian"
JOIN RecipeCategory in RecipeCategory TABLE
ON Recipes.Id equals RecipeCategory.RecipeId
WHERE RecipeCategory.CategoryId EQUALS "Pasta"

Neo4j Syntax

MATCH (Recipe)- [CONTAINS_MEAT]-> (Meat {"Bacon"}),
(Recipe)- [IS_FROM_CUISINE]-> (Cuisine {"Italian"}),
(Recipe)-[:IS_FROM_CATEGORY]-> (Category {"Pasta"})
return Distinct Recipe

Experiment number 3 represents an even more complex query defined on four tables (three joins) and two conditions. The results for experiment 3 appear in figure 6. The same observation can be made as in experiment 2. Neo4j executes the query much faster than MS SQL Server, and moreover, the execution time decreases drastically for the graph database system.



	Set 1	Set 2	Set 3	Set 4
■ SQL	3.1411699	5.5714943	10.5211421	16.6242377
■ Neo4j	0.519995	0.5560423	0.5364109	0.7268985

Fig. 6. Experiment 3 results

Experiment 4:

Query: Get all recipes similar to "Example Recipe" containing "Tomato"

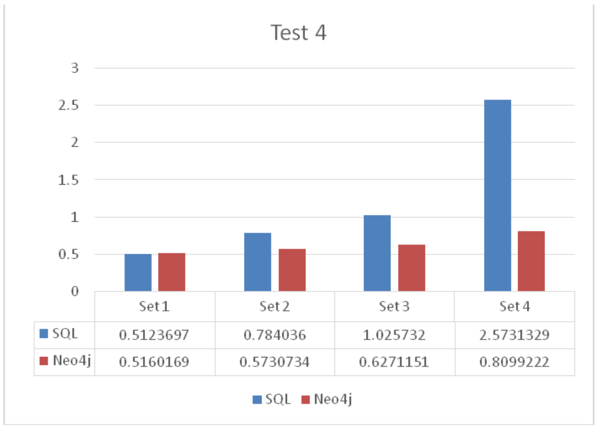
SQL Syntax

SELECT DISTINCT recipes
FROM Recipes IN Recipe TABLE
JOIN RecipesSimilarity IN RecipeRecipeSimilarity TABLE
ON Recipes.Id EQUALS RecipesSimilarity.RecipeTwoId
WHERE RecipesSimilarity.SimilarityId EQUALS "Strong" AND RecipesSimilarity.RecipeOneId EQUALS "Example Recipe" JOIN VegetableRecipe IN VegetableRecipe TABLE ON Recipes.Id EQUALS VegetableRecipe.RecipeId WHERE VegetableRecipe.VegetableId EQUALS "Tomato"

Neo4j Syntax

MATCH (Recipe {"Example Recipe"})- [similarity: IS_SIMILAR_TO {Similarity: "Strong"}]-> (Other Recipe),
(Other Recipe)-[CONTAINS_VEGETABLE]-> (Vegetable {"Tomato"})
return Distinct Other Recipe

The results for experiment 4 appear in figure 7. Again, Neo4J is superior to MS SQL server for all four data sets.



	Set 1	Set 2	Set 3	Set 4
■ SQL	0.5123697	0.784036	1.025732	2.5731329
■ Neo4j	0.5160169	0.5730734	0.6271151	0.8099222

Fig. 7. Experiment 4 results

Experiment 5:

Query: Get all recipes similar to "Example Recipe" that have more than 100 likes containing "Tomato" from the "Italian" or from the "Romanian" Cuisine

SQL Syntax

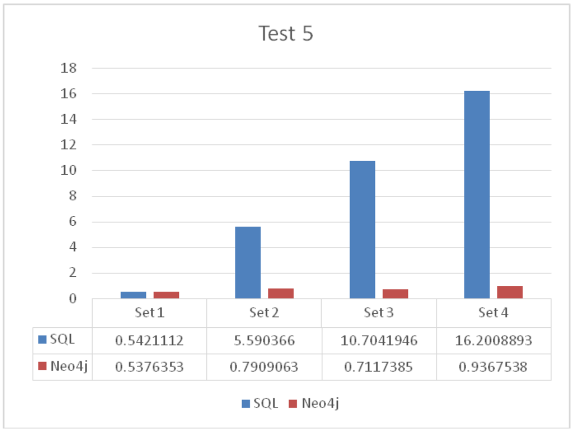
SELECT DISTINCT recipes
FROM Recipes IN Recipe TABLE
WHERE Recipes.Likes BIGGER THAN 100
JOIN RecipesSimilarity IN RecipeRecipeSimilarity TABLE
ON Recipes.Id EQUALS RecipesSimilarity.RecipeTwoId
WHERE RecipesSimilarity.SimilarityId EQUALS "Strong" AND RecipesSimilarity.RecipeOneId EQUALS "Example Recipe"
JOIN VegetableRecipe IN VegetableRecipe TABLE
ON Recipes.Id EQUALS VegetableRecipe.RecipeId
WHERE VegetableRecipe.VegetableId EQUALS "Tomato" JOIN RecipeCuisine in RecipeCuisine TABLE
ON Recipes.Id equals RecipeCuisine.RecipeId
WHERE RecipeCuisine.CuisineId EQUALS "Italian" OR RecipeCuisine.CuisineId EQUALS "Romanian"

Neo4j Syntax

MATCH (Recipe {"Example Recipe"})- [similarity: IS_SIMILAR_TO {Similarity: "Strong"}] -> (Other Recipe),
(Other Recipe)- [CONTAINS_VEGETABLE]-> (Vegetable {"Tomato"}),
(Other Recipe)- [IS FROM CUISINE]-> (Cuisine)
where Cuisine EQUALS "Italian" OR Cuisine EQUALS "Romanian" AND Other Recipe Likes BIGGER THAN 100
return Distinct Other Recipe

This query also involves four tables (three joins) and many conditions expressed with "and" or "or" operators.

The results for experiment 5 appear in figure 8. The same observation can be made. The query execution time that involves many junctions between tables and multiple conditions is much shorter in Neo4J than in the relational system and has also very little value for all four datasets.



	Set 1	Set 2	Set 3	Set 4
■ SQL	0.5421112	5.590366	10.7041946	16.2008893
■ Neo4j	0.5376353	0.7909063	0.7117385	0.9367538

Fig. 8. Experiment 5 results

V. CONCLUSIONS

The idea of this experimental study started as a Web Application that interacts with the end-user and quickly delivers responses based on the requests performed. However, nowadays, as the web applications are extensively popular and often used as opposed to physical items like books or magazines, they must adapt and be able to handle a large number of data. As the trend is to use SQL databases, that are the most popular databases worldwide, the development started with such a database as data storage 'device'.

When considering this from the future's perspective, it is interesting to analyze the manners in which they respond in such an application type, where there are many relationships between items and also, the recommending engine and how it will respond.

As described before, the same structure of the database was exported to a graph database, and for multiple sets of data, tests were performed. These tests were designed based on how users tend to interact with such a system.

As seen, for a slightly large number of records, where the query has to perform search based on a limited number of relationships SQL tend to perform better than the Neo4J database. However, as the numbers get bigger the Neo4J database appears to be superior when it comes to computing time. For a low number of JOINs, the SQL database doesn't fall back so much even with large numbers of records, but for this type of application, where items are strongly related through relationships the graph database, it is safe to say, it is clearly superior.

In conclusion, for applications that involve large number of relationships between data, the graph databases are a suitable choice. Such projects could be social media applications, collaborative systems or libraries of any kind, books, music or videos. Even though, relational databases are strong and well-performing, so, there are cases where there is a (slightly) better alternative for data storage.

Nowadays, many large companies world-wide have migrated to NoSQL alternatives and the results are astonishing. The performance of their applications is keeping users interested and satisfied everyday by providing fast responses to their requests and that is generating success.

When it comes to choosing what type of database should be used, one must first perform a type of research activity, read and most important perform tests on their applications ahead of time, with large numbers of records to predict how they will perform in the future.

Designing the application with a well-researched and well-chosen alternative is a critical step in the early stages of development. Performing changes along different development cycles and stages, when the application has already become complex, delivered and in use for users, implies migrating data from one database to another, which is a complex, time-consuming and high-risk task.

REFERENCES

[1] J. J. Miller, “Graph Database Applications and Concepts with Neo4j”, *in Proceedings of the Southern Association for Information Systems Conference*, Atlanta, GA, USA. Vol. 2324, No. 36, 2013

[2] http://neo4j.com/developer/cypher/

[3] https://linkurio.us/using-neo4j-to-build-a-recommendation-engine-based-on-collaborative-filtering/

[4] http://graphaware.com/neo4j/2013/10/11/neo4j-bidirectional-relationships.html

[5] https://neo4j.com/developer/graph-db-vs-rdbms/

[6] https://sdtimes.com/databases/guest-view-relational-vs-graph-databases-use/

[7] S. Medhi, and H. K. Baruah, “Relational Database And Graph Database: A Comparative Analysis”, New Technologies, International Vol. 5, No 2, 2017

[8] A. Martinez, R. Mora, D. Alvarado, G. Lopez, and S. Quiros, “ A Comparison between a Relational Databases and a Graph Database in the Context of a Personalized Cancer Treatment Application”, *in CEUR Workshop Proceedings*, Vol. 1644, 2016, http://ceur-ws.org/Vol-1644/paper37.pdf

[9] Y. Cheng, P. Ding,T. Wang, et al., “Which Category Is Better: Benchmarking Relational and Graph Database Management Systems”, *Data Sci. Eng.*, vol.4, pp. 309–322, 2019 https://doi.org/10.1007/s41019-019-00110-3

[10] W. Khan, E. Ahmed, and W. Shahzad, “Predictive Performance Comparison Analysis of Relational & NoSQL Graph Databases”, *International Journal of Advanced Computer Science and Applications*, vol. 8, no.5, 2017

[11] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, “Comparison between relational and NOSQL databases”, in *Proceedings of MIPRO*, pp. 0216-0221, 2018

[12] R. J. Sholichah, M. Imrona, and A. Alamsyah, “Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data”, in *Proceedings of ICITACEE*, pp. 152-157, 2020