# Comparison of SQL Relational Databases and NoSQL Graph Databases

Teodor Fratiloiu

***Abstract*—Classic, relational, SQL databases have proven their reliability, and versatility, in many different scenarios over the past few decades. However wide their field of application may be though, there are still many situations, where one can be limited by the constraints of these traditional data structures. Efficiently querying very large user databases is one such situation. In this work, we are going to investigate a different paradigm for big data management: Graph Databases. This relatively new breed of data management systems are already finding wide acceptance, and are being actively deployed, across many industries. Use cases such as social media applications come to mind, where scalability is of prime importance, since datasets could theoretically achieve infinite dimensions. In this work, we will compare Graph (NoSQL) databases, as well as their strengths, and weaknesses, to classic, relational SQL databases.**

**Keywords:** Relational Database, NoSQL Graph Database, Tablespace, Graph Mining

## I. Introduction

Databases have been used as the prime way of storing data in digital formats for the better part of the past 70 years years, since computer memory became large enough to enable this in the 1960's. It is safe to assume that virtually every kind of tabular data, which has since been digitalized, has been converted into one such databse, or a closely related approximation thereof. [4]

The history of databases has often been divided in 3 separate phases: firstly, there was the *classical* period of navigational databases (which strongly relied on the period's ever present memory pointers). This was followed by the invention of the relational (SQL) database by IBM's researchers in the early 1970's [5]. Lastly, around the turn of the millennia, researchers came up with the modern, post-SQL (or NoSQL) database, which has the distinction of having *no fixed structure*, or much in the way of theoretical constraints related to its overall architecture, if at all.

The reason for the existence of these new, very flexible types of databases has much to do with the main issue of classic, SQL databases. It is the exponential way in which hardware requirements scale up with the amount of data stored. This means that it becomes exponentially more challenging to maintain speed, RAM and storage amounts within reasonable constraints as the amount of stored data increases. This problem of SQL databases has been steadily developing over the decades, and it has to do with the difficulties that arise when mixing legacy database rules with the realities of modern, distributed and connected devices. [1]

Let us therefore introduce NoSQL databases. The main difference between these, and SQL databases, is how they scale. While SQL databases scale *vertically*, NoSQL databases scale *horizontally*. Specifically, it is the BASE (Basically Available State and Eventually Consistent) property of NoSQL databases which enables them to be a perfect choice for very large, connected, amounts of data.

For the purpose of this paper, we shall consider the term *relational* database (RDBMS) as synonymous to *SQL* database, and the term *NoSQL* database as equivalent to *Graph* database, unless other specified. As such, the respective pairs of terms will be used interchangeably.

It has been proved in research literature, as well as by the industry, that Graph databases perform radically better than relational databases for big data applications. [1] This is especially true in scenarios where one can use the connected nature of data to their advantage. One prominent example of this is how Neo4j (NoSQL) performs much better than Oracle 11g (SQL). [6]

However, let us not forget that Graph databases are not the only kind of NoSQL database out there. In fact, there are four types in total. Let us therefore examine these in detail.

### A. Types of NoSQL Databases

There are four different kinds of NoSQL databases. Their properties and use scenarios depend on the needs and requirements of the applications in which they are used.

- **Key-Value**: these are, essentially, hash tables, where only unique hash keys are used to search for and point to specific values; the contents themselves are not used during queries.
- **Document**: Data is stored as a key/value tuple, and both the content, as well as the *key* are used for searches, which constitutes a major difference from the type above. Documents of this type can be stored in XML, JSON and BSON formats, and popular apps using this type of database include MongoDB and CouchDB.
- **Wide-Column**: These databases are made unique by the hybrid approach used to store data. In this scenario, which is mainly used in *cluster environments* and for distributed data, we will combine the features of relational databases with a key-value storage schema. Examples such as Hbase, Cassandra and Accumulo spring to mind.
- **Graph**: This type is used where the data has a high degree of connectivity, since such databases can naturally store relationships between Independent pieces of data. Examples include Neo4j, Pregel, ArrangoDB and OrientDB.

Actual examples, together with actual package names, query languages and properties, can be found in *Table I*. [1]

Having now delivered our remarks about the fundamental differences between SQL and Graph Databases, let us proceed

TABLE I
TYPES OF NOSQL DATABASES WITH EXAMPLES

| Type | Package | Query Language | Property Support |
|---|---|---|---|
| Graph | Neo4j | CypherQL | ACID |
| Graph | ArangoDB | AQL | |
| | Hive | HQL | |
| Graph | OrientDB | SQL | ACID |
| Key-Value | Riak | | |
| Key-Value | Redis | | |
| Key-Value | Voldemort | | |
| Column | Hbase | | |
| Column | Accumulo | | |
| Document | MongoDB | | |
| Document | CouchDB | | |

with a more detailed description of the current state of development in the industry.

## II. STATE OF THE ART

In this section, we shall look into, and compare, the different properties of SQL and NoSQL databases. For starters, let us revisit the scalability issue. As previously discussed, the more varied and larger the amount of data present in a SQL database, the more CPU and RAM a system will need. This is because relational databases scale *vertically*, in contrast to NoSQL databases, which scale *horizontally*.

SQL databases are traditionally a good fit for applications where data integrity and security are important, since cryptographic procedures and information security can be more easily implemented in such a scenario. NoSQL databases, however, are a great fit for very large, and growing, datasets, because they allow us to, for instance, add extra storage servers to an existing system with minimal intervention. The following theoretical exercise might help one understand this better.

### A. The CAP Theorem

To enable us to better understand the fundamental differences between relational and Graph databses, let us introduce Eric Brewer's *CAP Theorem* - Consistency, Availability and Partition Tolerance. [7] A term by term analysis looks as follows: *Consistency* describes the property of a system of achieving a *stable state*, after one (or more) *write* operations. *Availability* describes data, namely it being present and ready for *read* operations at one desired spot, after having been modified (right there or somewhere else). *Partition Tolerance* refers to a system's ability to continue functioning, even if its resources are spread over different locations and nodes.

In relational databases, primary keys are used to ensure that data is *unique* - it exists in one single location. NoSQL databases, on the other hand, and in no small part due to their distributed nature, cannot provide such a feature. But let us not believe that has to necessarily be a bad thing. NoSQL databases will, because of this, be much more flexible and responsive to the changes one would experience in a connected system, for instance. Moreover, tracking interaction between different objects is much easier in this scenario.

However, we shall quickly realize that there is one obvious *catch* to all of this: a distributed database system can never offer true CAP features. Not in a logically rigorous manner, at least. That is because of the intrisic nature of how the data is stored in such a system cannot guarantee total and immediate consistency. Let us think about how data propagates across the internet. For the purpose of this, let one imagine they would change one's online profile picture for their Google profile, on their desktop. After the change is done, let one immeaditealy pick up their smartphone and open the YouTube app, where they had already been logged in with their Google account. One can very sensibly assume that the picture the user would see on their smartphone, at least for the first few moments, will be their old profile picture. This goes to prove that, for a brief period of time, that one user had *two different profile pictures*, at the same time. A state which very obviously contradicts any form of consistency.

To sum up, let one remember that relational databases traditionally possess all 3 CAP properties, whereas NoSQL databases (depending on the implementation) customarily only support the *Availability* and *Partition Tolerance* parts, but lack the *Consistency* feature.

### B. Relational Database Design Optimization

In this subsection, we shall make a small detour, and examine why tuning relational databases is so important, any why one should always consider doing it, both when deploying a product, and when running benchmarks on standard data to determine speed.

Let us look into how the performance of classic databases can be optimized. To this end, a process called *tuning* can be involved. For instance, to improve the query performance of *Oracle 11g* tables, we can use different tablespaces for each individual schema. This results in more than one data file used for storage, which in turn allows us to optimize the size of memory blocks used. This is useful, since both small and large memory block sizes can be desirable, depending on the scenario. For example, *warehouse type* data requires large memory blocks, whereas for *OLTP* type data, we are required to use small block sizes. The total space allocation in a file is defined as an *extent*, whereas one unit of I/O data is referred to as a *block*. The ACID (Atomicity, Consistency Isolation, Durability) property of SQL databases helps maintain consistency across the data, and enables efficient polling and querying, but even this is eventually overcome by the sheer size of modern databases for online, user-generated content.

Even if SQL databases store data in separate tables, and normalization techniques, such as 1NF, 2NF and 3NF, can be used to reduce redundancy, this method remains unsuitable for modern applications. For instance, although it is possible to *join* queries when searching a SQL databases for entries, this too becomes inefficient when using 20 or more parameters.

## III. METHODS AND CONTRIBUTIONS

For this section, we will investigate in more detail the structures of Graph and SQL databases, compare their respective performance, and present some interesting use cases for each.

### A. Data Mining and Grpah Visualization

One very important application of Graph Databases is their enabling more robust and faster data mining.

One author [**?**] describes the *Graph Mining Framework*.

### B. On SQL and the Importance of Information Security

## IV. Results

### A. Comparisons of SQL and NoSQL Products

In the following subsections, we shall analyze and compare several *competing* database types in a similar scenario.

1) *Comparing Oracle 11g to Neo4j:*
2) *Comparing Microsoft SQL to MongoDB:*
3) *Comparing MySQL to MongoDB:*
4) *Comparing ServerDB to RavenDB:*

## V. Conclusion

Having now accomplished what we set out to do - namely to present a thorough analysis, and comparison, of the two most common types of databases in use throughout industries in this day and age, let us now deliver the closing remarks and present future directions of research we consider relevant for undertaking.

Firstly, it is widely accepted that NoSQL, Graph databases are the desirable way to store data when systems are distributed and when flexibility, and redundancy, are needed. Such fields as online shops, social media, or some remote working tools rely on this type of system architectures, in order to ensure responsiveness, and reliable operation for multi-user collaboration. Further development and standardization of such systems could potentially open the doors to increased efficiency and reduced costs across the industry.

On the other hand, relational databases still play a very important role for centralized, high-value or mission-critical data. It is only with this type of database that one can ensure the reliable storage of such information as medical records, employment and financial histories, or voter registrations. Future research in this area could involve the development of even more sophisticated file systems, optimized for fast flash storage, or increasing information security by means of tightening standards for advanced encryption and by rolling out more and more granular access control.

Moreover, let us once more stress the fact that we do not assert the superiority of either database type over the other. We remain strongly convinced that both have their own strengths and weaknesses, which one only needs be aware of when making the choice about which best suits their needs. It is exactly for this reason that we presented the Hybrid approach to database design, in which many see an acceptable compromise.

Finally, after having illustrated the differing aspects of relational and NoSQL databases, and after having delivered our thoughts on the direction of future research, we hope to have at least awoken the interest for further research on the topic.

## References

[1] Wisal Khan, Waqas Ahmad, Bin Luo and Ejaz Ahmed, SQL Database with physical database tuning technique and NoSQL graph database comparisons (2019)

[2] Swapnil Shrivastava and Supriya N. Pal, Graph Mining Frameworks for Finding and Visualizing Substructures using Graph Databases (2009)

[3] H.R. Vyawahare, P.P. Karde and V.M. Thakare, A Hybrid Database Approach Using Graph and Relational Database (2019)

[4] Bachman, Charles W., The Programmer as Navigator. Communications of the ACM. 16 (11): 653–658. doi:10.1145/355611.362534. (1973)

[5] Codd, Edgar Frank, A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 13 (6): 377–387. doi:10.1145/362384.362685. S2CID 207549016 (June 1970)

[6] Oussous, A., F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, Comparison and Classication of NoSQL Databases for Big Data (in Proceedings of International Conference on Big Data, Cloud and Applications) (2015)

[7] Seth Gilbert and Nancy Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59. doi:10.1145/564585.564601.