Exploration and Presentation

Relational vs Non-Relational Databases

"To be or not to be a relational database? That is the question."

May 24, 2021

Authors
Jonas Valentin Björk Hein
Jonatan Magnus Klarskov Bakke
Thomas Stevns Nielsen Ebsen

Choosing the correct database for your application is a crucial element in almost every project that requires a datastore, and the fact that we collect more data than we have ever done in the past, emphasizes the importance of choosing the correct type of database structure. In this article we will be focusing on the difference between the rdbms and nosql models by explaining the use cases and how they perform when scaling.

1 Introduction

Relational Databases and NoSQL are two competing types of data models, and since the 1970's, the relational database model were the industries dominating storage method, up until the introduction of NoSQL in the early 2000, were NoSQL began replacing it in some applicational contexts [1]. The evolution of new technologies and the era of Web 2.0, changed the way we interact online, which triggered the need for a more versatile and scalable database model [2]. The introduction of NoSQL introduced a new way of dynamically storing data in a database, which led to the development of applications such as social media, e-commerce platforms, messaging software etc. where large amount of data are processed every second.

The shift from static to dynamic, caused many companies to switch part of their applications needs from the oled conventional method of using relational databases, to the more easily scalable and versatile type of database, like NoSQL. However, in spite of NoSQL being the new "trend" and applications becoming more and more dynamic, the relational database model are still widely used today, but why is that?

This paper is based on reviews and research of past literature and begins with a description of both relational and non-relational databases, including examples of use-cases for each model. The discussion then progresses to comparing the features between the two models and then concludes with a direct comparison, where we, based on our findings, attempt to determine wether NoSQL is a direct replacement for the relational-database model, and which one you should choose for your next project.

2 Relational and Non-Relational database models

The Rational and Non-Relational database models are two different approaches for storing volumes of data in a datastore. The Relational Database model is unique, and there is only one type of Relational Database [3], whereas NoSQL has multiple. There are many different types of NoSQL databases, but only four models are commonly used: *Document Databases, Key-value databases, Wide-column stores and Graph Databases* [4].

2.1 What describes a relational database

A relational database utilizes *tables, columns, rows* and *keys* to create structure and relationships [5]. If you wish to store information about certain people into the database, the first step would be to create a table which contain the persons information as *attributes* in the form of *columns*. Onces the *columns* are defined, they may be populated with a persons data, so when a new persons' information is added to the table, a row is created based on the columns.

id	firstname	lastname
1	Thomas	Ebsen
2	Jonas	Hein
3	Jonatan	Bakke

Figure 1: Figure of a Person Table in a relational database

This makes visualization of the saved data easier for the user to understand. The implementation of a primary key is also a requirement when creating tables and it acts as a rows unique identifier. Figure 1 displays how a database table would look if the primary key is implemented as an auto incremented integer, however it can be defined by any value as long as it is unique [5]. The primary key is crucial setting up relationships between tables, therefor it is important to choose the right unique identifier when defining it.

When creating tables, it is important to keep normalization in mind. Database Normalization is a technique used to organize and structure the information in the database, in accordance to the so called 'normal-forms'. When applying normalization, a systematic approach is taken, to split the data into tables, as to prevent data redundancy and repetition [6].

An example of applying database normalization can be as followed:

id	firstname	lastname	department	departmentAddress	departmentZip
1	Thomas	Ebsen	Department 1	Dep Add 1	6934
2	Jonas	Hein	Department 1	Dep Add 1	6934
3	Jonatan	Bakke	Department 2	Dep Add 2	3469

Figure 2: Figure of a Person Table in a relational database with no normalization

Figure 2 display a databases table with no normalization, and by storing data this way, we quickly run into a data redundancy issue, where Thomas and Jonas'

row contains some of the same data. By applying normalization as we explained above, the updated database tables should be structured like this:

People Table

id	firstname	lastname	departmentId
1	Thomas	Ebsen	1
2	Jonas	Hein	1
3	Jonatan	Bakke	2

Department Table

id	name	address	zip
1	Department 1	Dep add 1	6034
2	Department 2	Dep Add 2	3469

Figure 3: Applying normalization to Fig 2 to prevent redundancy

This cleared the issue of reocurring information and reduced redundancy by making the department information reusable.

Aside from Normalization, which is not an actual requirement, ACID is. All database that are built upon the Relational Database Model must adhere to the ACID principles. In layman's terms, ACID (Atomicity, Consistency, Isolation, Durability) references a set of standard properties that is used to guarantee that the database transactions are processed without issues [7].

As IBM explains it, the different ACID properties do the following [8]:

> *"In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.*
>
> - ***Atomicity**: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.*
> - ***Consistency**: Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.*
> - ***Isolation**: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.*

- ***Durability***: *After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.*

" - See [8].

By complying with the ACID properties, the DBMS (Database Management System) ensures the organization who employs the relational database model database strategy, that their database will maintain data accuracy and consistency, even if a failure occurs during the transaction process [7].

2.2 What describes a non-relational database

A Non-Relational database which is also called NoSQL or "not only SQL" [4], is a type of database that does not use the tabular form to store its data, but instead uses different data models, like the Document Database Model which we will use as a baseline for this section. In the Document Database Model, the storing of data differ from the standard tabular structure, in the way that it can store different types of data inside a document without validating the data, which means it is essentially possible to store any type of data into a document [9]. This type of storage method, is what makes it possible to store large and unstructured volumes of data easily, which means it is also much easier to import data sheets directly into the document, as it's setup is based on the data it received.

If you were to have an extensive CSV file filled with different information, that you wish to add into your database, it would prove difficult to do with traditional relational databases, since you would need to create tables for each different object in the csv and generate unique identifiers. With NoSQL, the task is relatively straight forward, since it's not bound by relational restrictions. By using NoSQL's document based model, we can save the CSV file directly into the database without any prior processing by the help of mongoimport [10]. The command will automatically convert it into a document, take the headers from the csv and convert them into value pairs and then save the data accordingly.

NoSQL is built upon the BASE model, which performs marginally better in instances where ACID would be excessive and hinder the useability of the database. Within BASE there are three principles [11].

- ***Basic Availability***: *The NoSQL database approach focuses on the availability of data even in the presence of multiple failures. It achieves this by using a highly distributed approach to database management. Instead of maintaining a single large data store and focusing on the fault tolerance of that store, NoSQL databases spread data across many storage systems with a high degree of replication. In the unlikely event that a failure disrupts access to a segment of data, this does not necessarily result in a complete database outage.*
- ***Soft State***: *BASE databases abandon the consistency requirements of the ACID model pretty much completely. One of the basic con-*

cepts behind BASE is that data consistency is the developer's problem and should not be handled by the database.

- ***Eventual Consistency**: The only requirement that NoSQL databases have regarding consistency is to require that at some point in the future, data will converge to a consistent state. No guarantees are made, however, about when this will occur. That is a complete departure from the immediate consistency requirement of ACID that prohibits a transaction from executing until the prior transaction has completed and the database has converged to a consistent state.*

" - See [8].

By complying with the BASE properties, the NoSQL database ensures the organization that their data will be highly available with eventual consistency.

3 Comparison

3.1 Where do relational databases perform well

After storing and categorizing the data into the tables, a view with complex sql queries can then be crated, to easily retrieve from or save to multiple tables at once.

Due to the nature of relational databases, navigating -and joining tables together is relatively uncomplicated, and complex queries are easy for users to carry out, which makes the structure flexible.

Multiple users can access, make changes and create requests simultaneously.

Relational database models are matured and therefore are well-understood adding to the trust behind them. Data in tables within a RDBMS can be restricted to allow access by given users.

By employing normalization, data is only stored once, which eliminates data duplication and redundancy [6].

3.2 Where do non-relational databases perform well

It can easily handle large amounts of complex and unstructured data, which makes the non-relational databases excel in scaling up existing databases. Non-relational databases are designed for horizontal scaling, this means that instead of scaling up by increasing the server's components like RAM, CPU, or SSD, non-relational database scales horizontally, so instead increasing a servers capacity, horizontal scaling is about adding multiple servers to handle the database [12]. This is due to a non-relational database not having the need to unpack the complexity of new data into tables, instead it stores the data into documents as we explained in section 2.2.

3.3 Where do relational databases fail to perform

Horizontal scaling is a huge downfall for relational databases. Relational databases are made to run on a single server maintaining the integrity of the table mappings. If you were to scale horizontal you would ruin the accuracy of the saved data. This is because when you have multiple instances of the database, syncing them will sometimes create inaccurate data across the instances, meaning that you might

oversee an update. With the design and properties of relational databases if the system needs to scale you need to scale it vertically to ensure accuracy. [13]

3.4 Where do non-relational databases fail to perform

A disadvantage when it comes to non-relational databases lies in not fully following the ACID properties. There are not many defined standards for NoSQL databases, which may result in inter-connectivity -and cross-integration issues between different NoSQL databases. [13]

This is where non-relational databases trade in performances, for the security that ACID brings when handling the data. The security that ACID brings lies in the way that it forces the database from handling new requests, until the data has hit the disk, which secures that data will not be lost in the process of handling large amounts of requests. Whereas non-relational databases they tend to follow BASE with stands for Basically Available, Soft State, and Eventually Consistent. BASE is all about performance and speed but lacks in the security departments this is because unlike ACID, BASE doesn't have to wait until the data have hit the disk, this makes a performance advantage but at the cost of risking to lose data you have just written [14] [15].

3.5 What are the differences

When it comes to comparing non-relational databases and relational-databases and when to use them, there will never be a clear cut answer to which one is best. This is of course because there will always be a reason to pick one above the other. Before distributed systems, the relational database with ACID properties did all what we needed, and at that time we only really had data stored in one place.

As technology grew we needed more scalability to deal with the massive amounts of data we wanted to store, and to scale a databases you will have to either go vertical or horizontal. Vertical scaling is harder to achieve because of high cost and cap to speeds on a single system. Horizontal scaling scales better since you have multiple databases running on separate systems. This is where non-relational databases really shine, because of how easy it is to save any kind of data set, and it's horizontal scaling [16]. Scaling is one of the big diverging features between non-relational and relational databases, but a big downfall for non-relational databases "take all" approach, is that it does not have a main query language unlike Relational Databases SQL. Relational databases are excellent at maintaining relationships and consistency due to of the table structure and ACID, whereas, non-relational databases are faster but lack consistency since they do not check if the write persisted which can result in loss of data.

4 Conclusion

Non-relational databases, are not a catch-all replacement for relational database and even though non-relational databases have a lot of good new use cases, there will always be a need for consistency and security when choosing a database. From reading our paper you should now have a broader understanding of the different database

structures and you have now gained more insight to which in the future should make your decision between relational or non-relational databases more streamlined.

References

[1] Keith D. Foote. *A brief history of Non-Relational databases*. URL: https://www.dataversity.net/a-brief-history-of-non-relational-databases/.

[2] Knut Haugen. *A Brief History of NoSQL*. URL: http://blog.knuthaugen.no/2010/03/a-brief-history-of-nosql.html.

[3] Mohammed O. Ismail Mohamed Ahmed Mohamed Obay G. Altrafi. *Relational Vs. NoSQL databases: A survey*. URL: https://www.researchgate.net/publication/263272704_Relational_Vs_NoSQL_databases_A_survey.

[4] MongoDB. *What is NoSQL?* URL: https://www.mongodb.com/nosql-explained.

[5] IBM Cloud Education. *What is a relational database?* URL: https://www.ibm.com/cloud/learn/relational-databases.

[6] Microsoft. *Description of the database normalization basics*. URL: https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description.

[7] Stephen Watts. *ACID Explained: Atomic, Consistent, Isolated and Durable*. URL: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable.

[8] IBM. *ACID properties of transactions*. URL: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions/.

[9] MongoDB. *What is a Non-Relational Database?* URL: https://www.mongodb.com/non-relational-database.

[10] MongoDB. *Mongo import examples*. URL: https://docs.mongodb.com/database-tools/mongoimport/#synopsis.

[11] Mike Chapple. *Abandoning ACID in Favor of BASE in Database Engineering*. URL: https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674.

[12] MongoDB. *Advantages of NoSQL Databases*. URL: https://www.mongodb.com/nosql-explained/advantages.

[13] Team LoginRadius. *RDBMS vs NoSQL*. URL: https://www.loginradius.com/blog/async/relational-database-management-system-rdbms-vs-nosql/.

[14] Bryce Merkl Sasaki. *Graph Databases for Beginners: ACID vs. BASE Explained*. URL: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/.

[15] Charles Roe. *ACID vs. BASE: The Shifting pH of Database Transaction Processing*. URL: https://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/.

[16] Evan Tarver. *Horizontal vs. Vertical Integration: What's the Difference?* URL: https://www.investopedia.com/ask/answers/051315/what-difference-between-horizontal-integration-and-vertical-integration.asp/.