Use of a NoSQL database to improve accountability

A case study for the Dutch tax authorities

LAB University of Applied Sciences
Bachelor of Information Technologies
2021
Emmi Moilanen

Abstract

Author(s) Moilanen, Emmi	Publication type Thesis, UAS	Published 2021
	Number of pages 33	
Title of Publication **Use of a NoSQL database to improve accountability** A case study for the Dutch tax authorities		
Degree and field of study Bachelor of Information Technologies		
Name, title and organisation of the client (if the thesis work is commissioned by another party) Kyra van Onselen, PhD Sc., Dutch Tax Office		

Abstract

The Dutch tax office faces two major problems. Firstly they have to be able to store large amounts of information while having to comply with various laws which impose requirements on the management of the information. Secondly, over the years it has proven difficult to provide specific in-depth information when the Dutch Parliament wanted to be better informed on a specific subject. As a solution, the use of a NoSQL type of database is considered.

This thesis gives insight into the advantages and disadvantages of the four most commonly used NoSQL databases (MongoDB, Redis, Cassandra and Neo4j). In addition results of practical tests with two Document store databases technologies are presented (RavenDB and MongoDB). Based on these results only general conclusions could be made:

- Using a NoSQL database is a feasible option for the Dutch tax office
- Of the four types of NoSQL solutions investigated, a document style database seems to fit the requirements of the Dutch Tax office best.

Keywords

NoSQL, DMS, databases

Contents

1 Introduction

To prevent fraudulent use of childcare allowances, the government of the Netherlands has applied very strict rules regarding the benefits system during the last decade. These policies have, however, led to highly undesirable situations for the individuals and families who are dependent on the said allowances. In some cases, people receiving benefits have been wrongly flagged as fraudsters and have consequently had to pay back enormous amounts of childcare allowances. As a result, people have been driven to significant financial problems (Inspectie Overheidsinformatie en Erfgoed 2021). The Dutch government had to resign due to this chain of events (Roobeek, Frater, & Kennedy,2021).

A parliamentary committee of inquiry has investigated how this situation with the childcare allowances could have arisen and why the government has not intervened earlier (Tweede Kamer der Staten-Generaal, 2020). One of their conclusions was that the information management system currently used by the tax office, which is in charge of handling the benefits and possible benefit withdrawals, is not up to modern standards (Rutte, 2021). The House of Representatives has agreed that not only the information management system has to be improved, but significantly more information has to be recorded and archived (Snels & Van Weyenberg, 2021).

1.1 Objective and delimitations

On the one hand, the Dutch tax office has to be able to store huge amounts of data without adding too much additional workload for its employees, while still being able to quickly come up with information on specific topics when asked for. As a solution for this paradox, the use of a NoSQL type of database is considered.

The objective of this study is to advise the Dutch Tax office on which type of NoSQL database best suits their needs.

This study will only look at the usability aspect of this matter, such as the balance between the effort needed to store information and perform queries on all kinds of topics. For the tax authorities to make an informed choice, other aspects have to be considered as well. For example, the purchase and maintenance costs of these types of databases. This study aims to provide the ‘technical part’ for the business model the tax office will have to make.

1.2 Research questions

To be able to give sound advice, the following research questions have to be answered:

1. What type of NoSQL databases are available and what are their main characteristics?
2. What are key performance indicators for the Dutch tax office?
3. How well do the different types of databases perform on these key performance indicators?

1.3 Research Method

The main objective of this study is to advise the Dutch Tax office on which type of NoSQL database would best fit their key performance indicators on information management. To fulfil this objective, a literature study has been performed and subsequently, the knowledge gained is applied to the use case of the Dutch tax authorities. Finding the answers to the above research questions results in qualitative empirical data. Therefore, a qualitative research method has been used.

1.3.1 Data acquisition and analysis

The following information is needed to answer the research questions as outlined for this study:

- What kind of information should be archived?
- How much information should be archived?
- How long does information need to be stored?
- What kind of inquiries are expected?
- How much effort should go into archiving (eg. adding metadata) versus how much effort can be spent in retrieving the information?
- Is data retrieval sufficient or should it be possible to perform some basic management related functions as well (eg. combining monthly reports into a yearly one, averaging weekly data)

The necessary empirical data is acquired from already available information such as reports, websites, etc. In addition to these sources, an interview has been conducted with some employees of the Dutch tax office.

1.3.2 Methodological literature

The theoretical framework is mostly based on methodological literature. Since all libraries were closed at the moment the theoretical part of this study was performed, only digitally available literature could be used.

1.4 Outline

The structure of this study is composed of five main chapters. The second main chapter contains the theoretical framework of the study. This theoretical chapter includes explanations of document management systems and database solutions, giving further insight into the advantages and disadvantages of using either a relational database or NoSQL database solutions for the DMS.

Chapter four will provide a better understanding of the need to improve the information management system and the problems involved. This chapter provides more detailed insight into the Dutch government information policy and how these policies might have changed. An overview of the applicable laws and regulations, and how information is currently managed by the Dutch government will also be given in this chapter.

In the following sections, the three research questions will be addressed. In chapter 3 a theoretical framework on databases will be given. In addition, the four most commonly used NoSQL databases (MongoDB, Redis, Cassandra and Neo4j) will be compared. This section aims to provide insight into the advantages and disadvantages of these four types of NoSQL database solutions.

To get a sense of what it's like to work with a NoSQL database some practical tests have been performed as well. For these tests two Document store databases technologies have been chosen RavenDB and MongoDB). This choice was motivated by the fact that these two databases offered the most user-friendly non-subscription trial clients. The results of these tests are given in chapter 4.

To be able to evaluate the different types of databases on how well they might work for the Dutch Tax office key performance indicators are needed. However, at this moment the Tax office has not yet mapped out the functional demands for their new DMS. Therefore, these functional demands, and subsequently the key performance indicators, needed to be derived from (a.o) an overview of the problems the Tax office are facing. The results are presented in chapter 2.

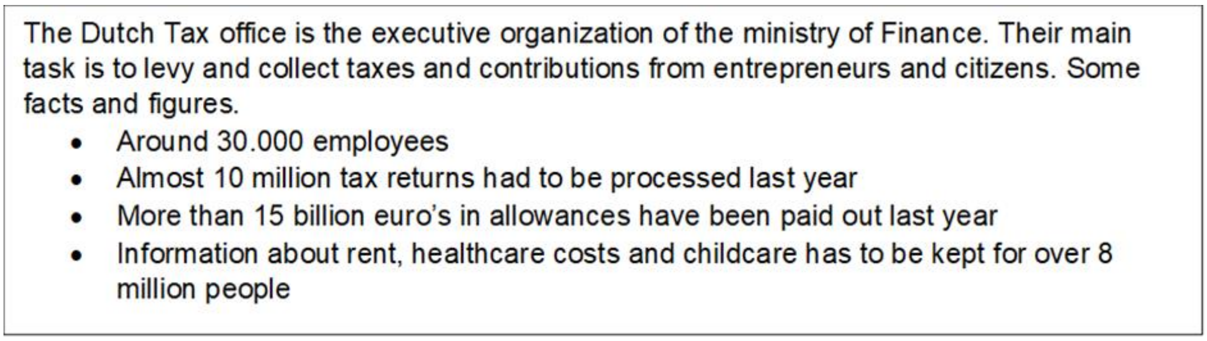
Conclusions concerning the objective of this study and the three research questions are given in chapter 5. In this chapter also some recommendations are made for further research.

2 Key performance indicators of the Dutch tax office

There is a wide range of possibilities in studying databases. However, this study does not aim at giving a complete overview of (and comparison between) all kind of database types and their underlying infrastructure. Instead, the aim is to focus on providing well-founded advice on the merits of the different types of NoSQL databases for the Dutch tax office. Therefore, this theoretical framework is limited to the specific functional demands of the case company.

2.1 Functional demands

Within the Tax office, there are several different systems including local and collective computer directories, various dedicated databases and physical archives. All these systems have to be combined into one document management system (DMS). At this moment, no decision has been made on the functional demands for this DMS and therefore, functional demand for the underlying database is not yet available. As the first step to elaborate functional demands for this study, some facts and figures about the Tax office have been collected; see Figure 1 (K. van Onselen, personal communication, February 22, 2021).

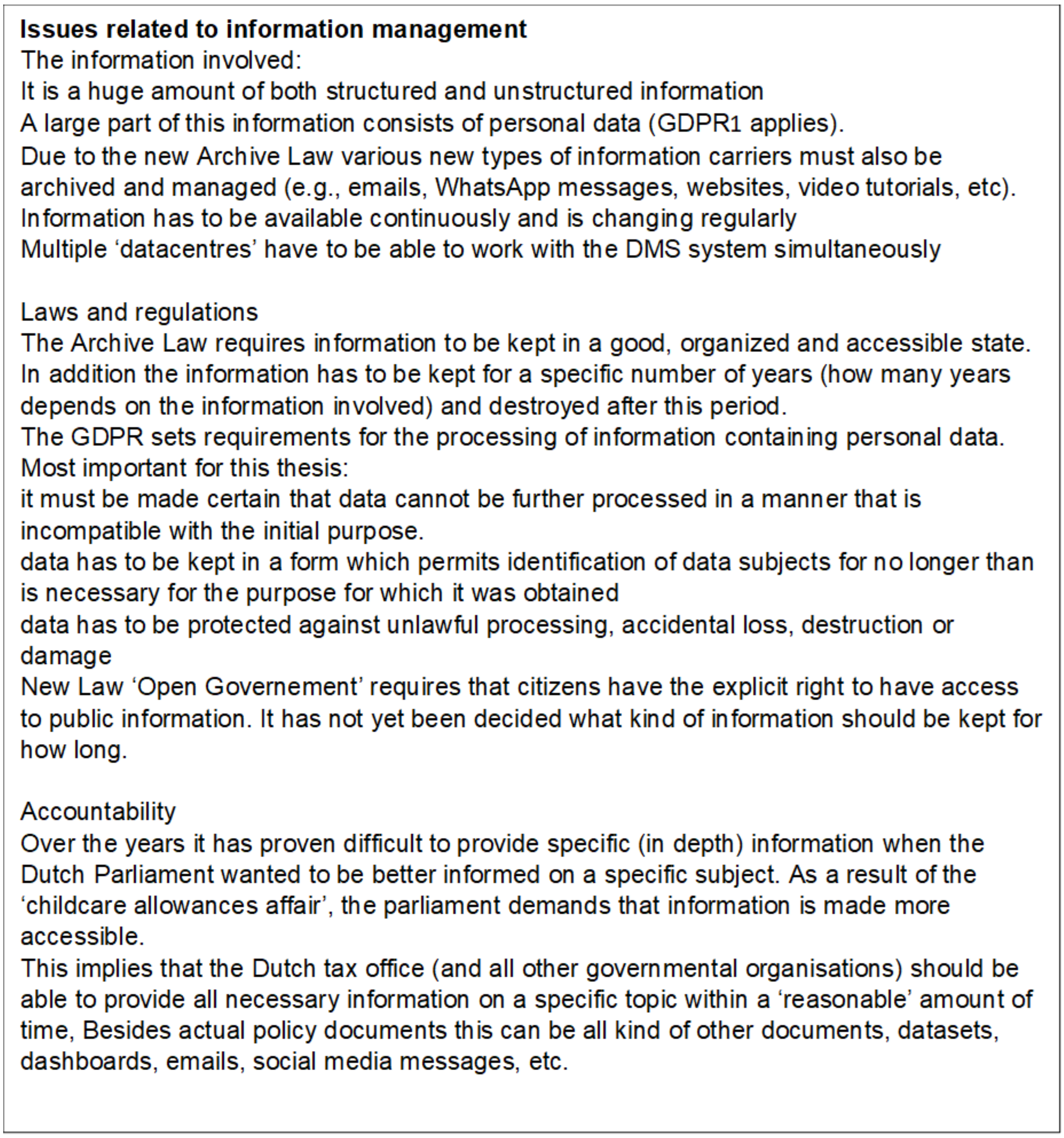


The Dutch Tax office is the executive organization of the ministry of Finance. Their main task is to levy and collect taxes and contributions from entrepreneurs and citizens. Some facts and figures.

- Around 30.000 employees
- Almost 10 million tax returns had to be processed last year
- More than 15 billion euro's in allowances have been paid out last year
- Information about rent, healthcare costs and childcare has to be kept for over 8 million people

Figure 1 Some facts and figures about the Dutch Tax office (K. van Onselen, personal communication, February 22, 2021)

The Dutch Tax office tax authorities need a significant amount of information to carry out their day-to-day activities. This information must be correct, reliable and easily accessible (Belastingdienst, personal communication, June 7, 2021). As a second step to generate functional demands the issues related to information management have been identified. They are presented in Figure 2.



Issues related to information management
The information involved:
It is a huge amount of both structured and unstructured information
A large part of this information consists of personal data (GDPR1 applies).
Due to the new Archive Law various new types of information carriers must also be archived and managed (e.g., emails, WhatsApp messages, websites, video tutorials, etc).
Information has to be available continuously and is changing regularly
Multiple 'datacentres' have to be able to work with the DMS system simultaneously

Laws and regulations
The Archive Law requires information to be kept in a good, organized and accessible state. In addition the information has to be kept for a specific number of years (how many years depends on the information involved) and destroyed after this period.
The GDPR sets requirements for the processing of information containing personal data. Most important for this thesis:
it must be made certain that data cannot be further processed in a manner that is incompatible with the initial purpose.
data has to be kept in a form which permits identification of data subjects for no longer than is necessary for the purpose for which it was obtained
data has to be protected against unlawful processing, accidental loss, destruction or damage
New Law 'Open Governement' requires that citizens have the explicit right to have access to public information. It has not yet been decided what kind of information should be kept for how long.

Accountability
Over the years it has proven difficult to provide specific (in depth) information when the Dutch Parliament wanted to be better informed on a specific subject. As a result of the 'childcare allowances affair', the parliament demands that information is made more accessible.
This implies that the Dutch tax office (and all other governmental organisations) should be able to provide all necessary information on a specific topic within a 'reasonable' amount of time, Besides actual policy documents this can be all kind of other documents, datasets, dashboards, emails, social media messages, etc.

Figure 2 Issues faced by the Dutch Tax office related to information management (K. van Onselen, personal communication, February 22, 2021).

2.2 Key performance indicators

Before the tax authorities can implement a new DMS, the technical, financial, organizational, legal and administrative feasibility must first be investigated. This thesis focuses on the technical feasibility of using a NoSQL database. In particular different types of NoSQL databases have been looked at to gain insight into which database solution fits the functional requirements of the Tax office best. To compare the different databases key performance indicators are needed. In Table 1 the indicators are shown that have been derived based on the information presented in paragraph 3.1.

Table 1. Key performance indicators for comparing the NoSQL databases.

Key performance indicator	**Issues**
Usability	Is it possible to store all information types needed?
	Can it handle the type of queries that are needed for accountability?
	Can the data be protected?
	How much effort is storing data and making queries?
Performance	Is information continuously available?
	Can information be updated regularly?
	Can multiple data centres work with the same database?
	Is the data protected against corruption?
Flexibility	Can the database easily be expanded (size, type of information)
	Is it easy to incorporate new types of queries?
	to what extent is it possible to make use of newly developed 'underlying' technology

3 Theoretical framework on databases

3.1 Introduction to databases

The database is at its simplest meaning a collection or assortment of data that is structured in a way that it's conveniently stored and retrieved in an electrical form - an integrated collection of records. Databases are structured in a way that the stored data is connected via some kind of relationship or by using additional information – metadata – most commonly defined as data about data. (Duval, 2001, p. 1)

Around the 1960s modernization of private companies led to the need for more effective data management systems. During this period, two widely used database systems were developed; i.e. IMS (Information Management System) and CODASYL (Conference On Data System Language). IMS is a hierarchical model that was developed by IBM as a response to the rapidly increasing need to store large amounts of detailed and integral data by parties like NASA (IBM, n.d. a).

Even though IBM would later develop a more modern relational database known as IBM DB2, IMS still remains a popular option for companies running legacy database systems.

CODASYL is an organization of volunteer representatives of computer manufacturers and users. They developed a network database Management System (DBMS) whose set of specifications are still partly used in all DMBS designed afterwards (Burleson Consulting, 1996).

3.2 Relational databases

A major disadvantage of the early database models developed in the 1960s was that retrieving data was often tedious and difficult. In 1970 Edgar Codd developed the first relational database model which made them more accessible for people without specific computer knowledge (IBM, n.d. b).

Codd (1970) used the term relational to indicate that there was a relation between groups of datasets. By defining these relations users could retrieve information from the database without needing specific knowledge of the physical structure of the database.

Nowadays the market offers a wide range of relational database technologies. The website DB-engines (2021 a) lists a total of 152 relational database technologies. According to them, the most popular are Oracle, MySQL, Microsoft SQL Server, PostgreSQL and IBM Db2. These databases are all based on (a part of) Codd's 12 rules (Codd 1970). Therefore, Graham (2009, s12) summarizes that data is organized as relations, attributes and domains.

A relation refers to a table with rows and columns (tuples and attributes). Furthermore, each row is unique and can only contain a single value for all of its set attributes. Finally, the domain specifies which values attributes are allowed to use. This is illustrated by Figure 3

Term	Description
Tuple	A data set representing a single item
attribute	A labeled element of a tuple, e.g. "Address" or "Date of birth"
Relation	A set of tuples sharing the same attributes; a set of columns and rows

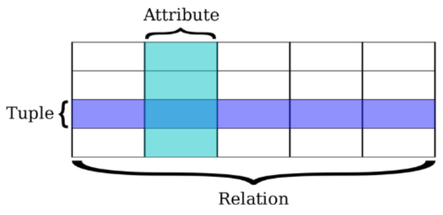


Figure 3. The general structure of relational databases. Based on Wikipedia (n.d.)

3.3 NoSQL databases

Relational databases were for the last thirty years considered to be the standard database for managing data. However, the development of modern applications (web 2.0/3.0, big data, etc) has given rise to the use of NoSQL databases (Kunda and Phiri, 2017).

The term NoSQL was first introduced by Carlo Strozzi in 1998 He used this term to indicate that this new technology was not an SQL database but rather a shell-level tool (Strozzi, 2007). Nowadays NoSQL refers to ‘Not Only SQL’.

During the last decades, different types of NoSQL databases have been developed. They can be divided into four categories (Zollman, 2012):

- Key-Value stores.

  They are associative arrays consisting of values and keys. Each key has to be unique to provide non-ambiguous identification of values.

- Wide column stores.

  An extended version of a Key-Value store that uses a two-dimensional key. Consequently, a stored value needs both a row key and a column key. Adding of even more keys is supported; e.g. using a timestamp

- Graph databases

  Data is represented by graphs. A major advantage is that these graphs allow for working with a flexible and very high number of interconnections.

- Document stores

Data is stored in a structured format. The structure is flexible but a specific data format has to be chosen.

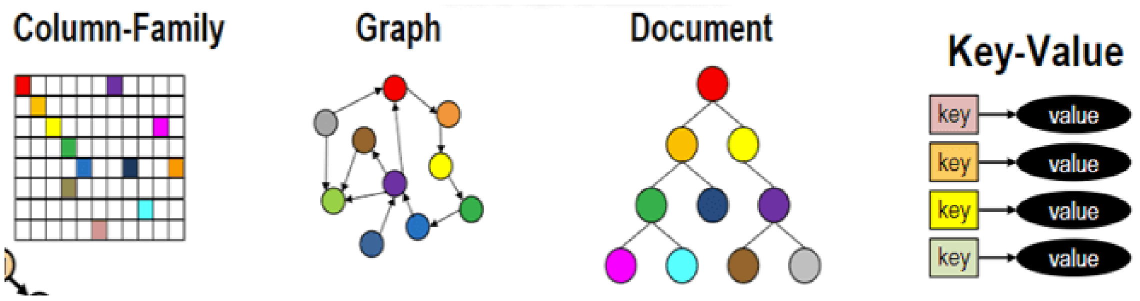


Figure 4. Four categories of NoSQL databases. Copied from Foote (2018)

To illustrate the different categories of NoSQL databases Figure 4 shows how data is referred to in these different types of models. An example of how data is represented in these four types of databases is shown in Figure 5.

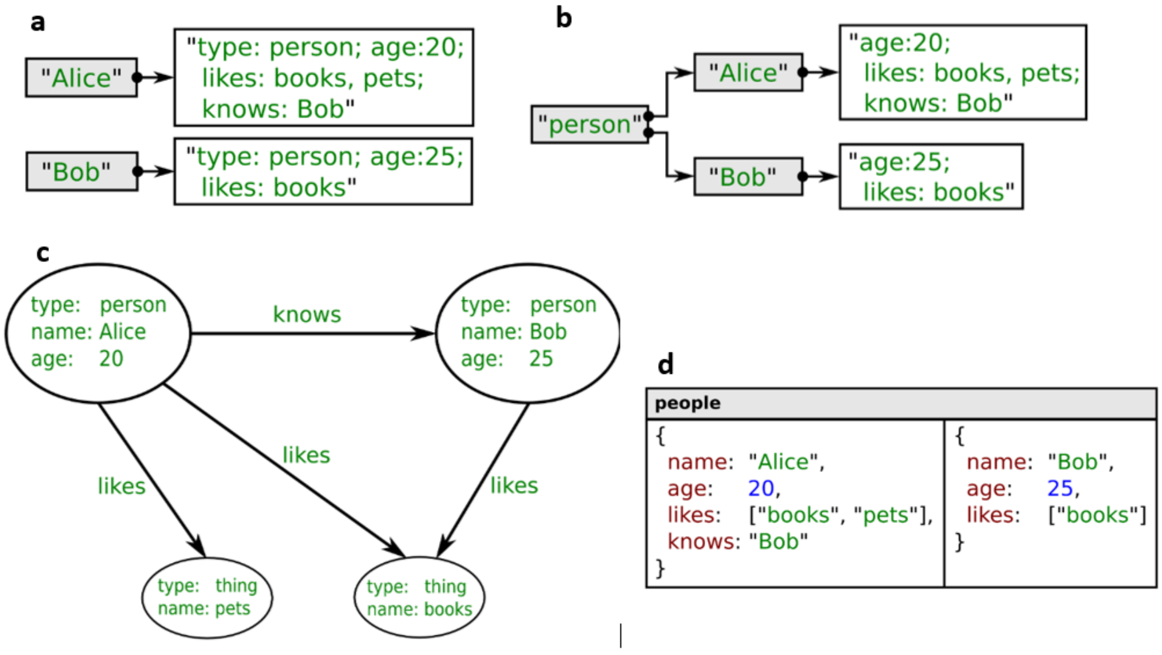

a
"Alice"
"type: person; age:20; likes: books, pets; knows: Bob"
"Bob"
"type: person; age:25; likes: books"
b
"person"
"Alice"
"age:20; likes: books, pets; knows: Bob"
"Bob"
"age:25; likes: books"
c
type: person name: Alice age: 20
knows
type: person name: Bob age: 25
likes
likes
likes
type: thing name: pets
type: thing name: books
d
people
{ name: "Alice", age: 20, likes: ["books", "pets"], knows: "Bob" }
{ name: "Bob", age: 25, likes: ["books"] }

Figure 5 Example data represented in the four categories of NoSQL databases. a) Key-Value store, b) Wide column store, c) graph database, d) document store. Based on Zollman (2012).

Using NoSQL databases has several advantages. NoSQL databases allow for (Radoev, 2017):

- The storage and processing of large volumes of data
- Storing both structured and unstructured data
- Flexibility (e.g. adapting the structure of the data)

- Scalability (e.g., extend the scale of stored data and applications).

Nowadays, a wide range of relational databases is available. DB-engines lists a total of 64 Key-Value stores (DB-engines, 2021 b), 13 Wide Column stores (DB-engines, 2021 c), 36 Graph databases (DB-engines, 2021 d), 53 Document stores (DB-engines, 2021 e). According to them, the most popular are MongoDB (document), Redis (Key Value), Cassandra (Wide Column) and Neo4j (Graph).

3.3.1 Quality parameters for evaluating NoSQL databases

To compare the performance of (among others) the various available NoSQL database the "Yahoo! Cloud Serving Benchmark" (YCSB) framework was developed (Cooper et al., 2010). Performance indicators in this benchmark are read/write, latency and elasticity capabilities. In their studies, Lourenço et al. (2015) have added the quality attributes consistency, robustness and maintainability to better evaluate NoSQL databases.

Online, a significant amount of technical articles can be found which compare and/or test NoSQL databases. Unfortunately, these gave only partial insight into what type of NoSQL database would be advisable for the Dutch Tax Office. This is caused by the fact that these articles:

- Have been written for specific applications like geospatial workloads (Kim and Kanwar, 2019) or big data use cases (Endres et al. 2020).
- Cover only part of the functional demand (Flores et al., 2018 about evaluation of NoSQL queries in response time for E-government), (Nurhadi et al., 2020 about Smart City Data Lake management).
- Are probably too old to be completely useful (Lourenço et al., 2015)
- Often have only an abstract that is free of charge and obtaining the complete article is costly.

As explained above, the conducted desk study into the four most popular NoSQL databases (MongoDB, Redis, Cassandra and Neo4j) didn’t give sufficient substantiation for advising the Dutch Tax office. Therefore, in the following paragraphs, only the (major) advantages and disadvantages of these databases are listed. In addition to the desk study, a practical test has been performed with two Document stores (MongoDB and RavenDB); see chapter 4. Paragraph 3.3.6 contains a theoretical comparison between these two types of document stores.

3.3.2 Evaluation of MongoDB

MongoDB is an open-source document store NoSQL database. According to Zollmann (2012), it was developed to handle large amounts of data. Since it is a document-oriented database, data is organized as a collection of documents with possibly different structures (Krstić, 2018). According to Krstić (2018) document stores are the most popular type of NoSQL database because of their flexibility, performance, and ease of use.

Document stores are especially suitable for (Ploetz et al., 2018), (Krstić, 2018):

- Storing large amounts of data
- Working with quick-changing data
- Regularly adding new data groups
- Filtering documents based on attributes in the document itself.
- Tracking changeable metadata types

In Table 2 some advantages and disadvantages are listed of the MongoDB document store. These are based on the articles by Ploetz et al. (2018) and Krstić (2018).

Advantages of MongoDB	**Disadvantages of MongoDB**
It is very simple to work with (a.o. easy environment and quick set-up)	Limited data size 16 MB for a document) and nesting (100 levels)
Complex joins are not required	High memory usage
It is easy to scale	Transactions may lead to corruption of data
It is flexible	It doesn't support the joining of documents
It uses sharding while handling large datasets	The domain of application is limited
Conversion or mapping between database objects and application objects is simple	Some applications do not allow ad hoc querying and changing data
Fast access of data is achieved by integrated memory support	some implementations do not stand high-frequency data changes
It has rich query support	
Semi-automatic replication and horizontal collection partition are supported.	
Documents inserted in the collections can have different sets of fields	

Table 2 Advantages and disadvantages of MongoDB. Based on Ploetz et al. (2018), Krstić (2018), Knowledgenile (n.d. a)

3.3.3 Evaluation of Redis

Redis is an open-source, in-memory data structure store, used as a database, cache, and message broker (Redis, n.d.). It provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, geospatial indexes, and streams. According to Krstić (2018), a Key Store database like Redis can be seen as a table in a relational database with two columns (the key and the value).

Key Store databases in general, and Redis, in particular, are useful for (Krstić, 2018), (Knowledgenile (n.d. b):

- Situations in which simple data models can be used and applications have frequent short readings
- Storing huge amounts of data while maintaining fast responses and with relatively low risks of downtimes
- Monitoring temporary attributes in a Web application
- Storing configurations and user information for mobile applications,
- Storing large objects (e.g. audio files and images)

Table 3 Advantages and disadvantages of Redis. Based on Krstić (2018) and Weekly Webtips (2020).

Advantages of Redis	**Disadvantages of Redis**
It is fast	It has a high level of redundancy
It is easy to set-up	Larger numbers of collections lead to more complex structures
It can work with huge amounts of data	Huge decrease in efficiency when data is 'densely' linked
It has flexible data structures and almost all types of data structures can be used.	There is no mechanism for ensuring the integrity
Keys and value pairs can be as large as 512 MB	The search condition is limited to a fixed key value or a range of key values.
It has its own hashing machine	Data is sharded based on the hash-slots assigned to each Master. If Master holding some slots is down, data to be written to that slot will be lost.
Scaling down doesn't cause downtime and has no performance impact	A huge ram is needed
Semi-automatic replication and horizontal collection partition are supported.	

3.3.4 Evaluation of Cassandra

According Appache (n.d.) Cassandra is "an open-source NoSQL distributed database trusted by thousands of companies for scalability and high availability without compromising performance. the perfect platform for mission-critical data". Review sites score Cassandra an average of seven (out of 10), but they also show that a lot of reviewers prefer other NoSQL solutions (TrustRadius, n.d.), (Capterra, n.d.).

Table 4 Advantages and disadvantages of Cassandra. Based on Krstić (2018), TrustRadius (n.d.) and Capterra n.d.).

Advantages of Cassandra	**Disadvantages of Cassandra**
It can handle large amounts of data	It has a high level of redundancy
It is fast (except in some cases of values with extremely complex structures)	Larger numbers of collections lead to more complex structures
It can work with huge amounts of data	Huge decrease in efficiency when data is 'densely' linked
It is great for distributed data design	There is no mechanism for ensuring the integrity
It allows for operating from multiple datacenters with little or no data loss	The search condition is limited to a fixed key value or a range of key values.
It has continuous data availability	Problems can arise with querying on a large amount of data.
Automatic replication and horizontal collection partition are supported.	Scalability is difficult

3.3.5 Evaluation of Neo4j

Neo4j is an open-source, distributed data store used to model graph problems. According to Ploetz et al. (2018) Neo4j stores information in schema-less, entity-like structures which are called nodes. These nodes are connected to other nodes via relationships or edges. It is also possible to group nodes together with optional structures, or so-called labels. According to Krstić (2018) Graph databases are suitable for applications in which data solving and storage can be done using graphical structures. Graph databases are the best solution for handling connected data and are optimized for managing relations (RubyGarage, n.d.). This makes this type of database convenient to use for, amongst others:

- identity and access management
- IT infrastructure management
- Fraud detection and analytics

- Recommendation of specific products based on functional demands
- Privacy and risk compliance

Table 5 Advantages and disadvantages of Neo4j. Based on Krstić (2018) and RudyGarage (n.d.)

Advantages of Neo4j	**Disadvantages of Neo4j**
They are very effective in the case of common operations with graphs	The application domain is restricted
The structure and schema of a graph model can be easily adjusted to the changes in an application	
The data structure can be easily upgraded without damaging existing functionality.	
The structure of the database can be easily upgraded. Therefore the data store can evolve along with your application.	

3.3.6 Comparison between MongoDB and RavenDB

According to RavenDB (2019), RavenDB is a document database built for fast performance, minimal complexity, short release cycles and little to no need for support. In Table 2 some aspects are compared for RavenDB and MongoDB (Lavi, 2021), Warda (2019). This comparison is only meant to give some insight into the differences between these two databases. To make a decision on which database is most suitable for the Dutch Tax office all aspects should be taken into account. Warda (2019) gives an in-depth analysis of the differences between the two Document Style databases. However, it is beyond the scope of this thesis to make the necessary theoretical analysis to use his results. Therefore, some practical tests have been performed with both RavenDB and MongoDB; see chapter 4 for the results.

Table 6 Global comparison between MongoDB and RavenDB. Based on Lavi (2021) and Warda (2019)

Aspect	**Comparison between MongoDB & RavenDB**
total implementation cost (TCI)	TCI of RavenDB is higher
the total cost of ownership (TCO)	TCO of RavenDB is higher
Available features	They both have Data Import/Export, Basic Reports, Online Customer Support
Target customer size	MongoDB: medium business, Large business, Private use RavenDB: Small, medium and large size businesses.
Learning to do queries	MongoDB doesn’t use SQL like syntax for query operations. Therefore, a significant amount of time has to be invested in learning to do this. RavenDB query syntax is SQL-like.

4 Practical comparison between different types of NoSQL databases

To advise the Dutch Tax office on which type of NoSQL database would best fit their key performance indicators on information management both theoretical and practical studies have been performed. As presented in section 3, the four different types of database solutions have been included in the theoretical studies. Ideally, all these types of databases would have been tested as well. However, because of several practical reasons, the possibility for a practical comparison is limited:

- Because the major part of the information the Tax offices is confidential, it was not possible to work with real data.
- Only non-subscription, free database trials could be used. This comes with the disadvantage of the available features being very limited.
- The scope of a bachelor thesis doesn't allow for building large databases and developing complex queries.

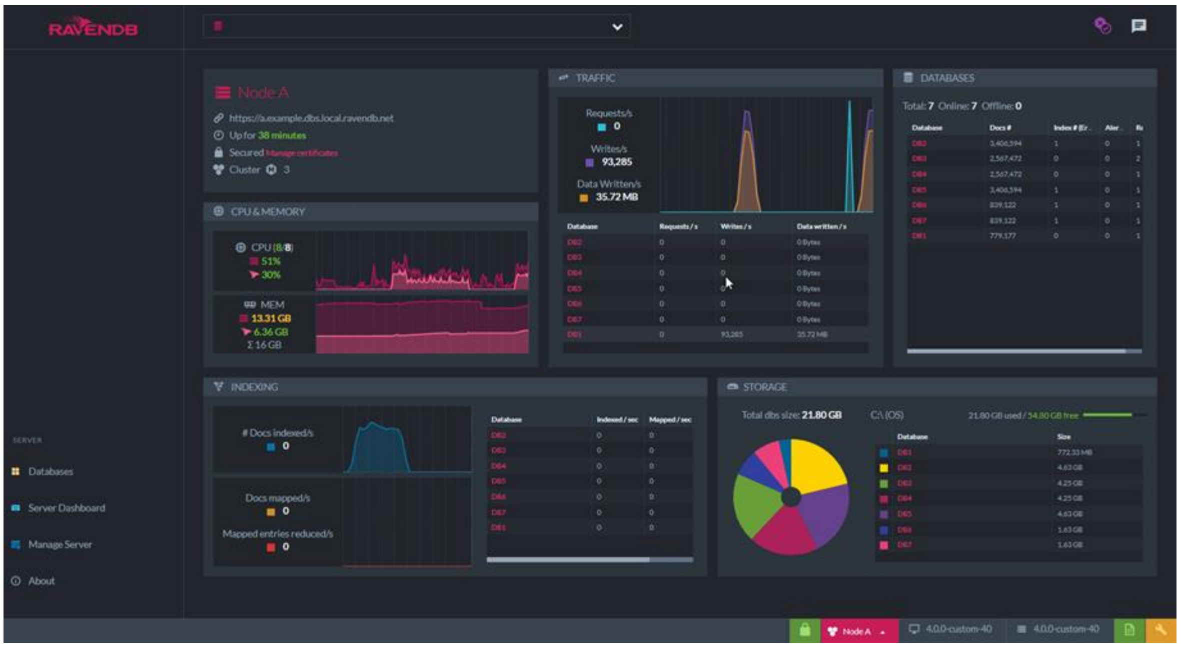
Because of the limitations mentioned above, the practical test has been limited to

- Testing only two database solutions
- Setting up the database environment
- Populating the test database with JSON data documents
- Running simple queries

RavenDB and MongoDB have been selected for performing the practical experiment. From the available options, they offered the most user-friendly non-subscription trial clients.

4.1 RavenDB

RavenDB offers both commercial and non-commercial licenses. The commercial license is only available as a yearly subscription. Therefore, the non-commercial license, which is free of charge, has been used for this practical comparison. When considering the results of this test, it has to be kept in mind that this non-commercial license offers significantly fewer features than the commercial one which the tax office would need to have.



Database	Requests / s	Writes / s	Data written / s
DB2	0	0	0 Bytes
DB3	0	0	0 Bytes
DB4	0	0	0 Bytes
DB5	0	0	0 Bytes
DB6	0	0	0 Bytes
DB7	0	0	0 Bytes
DB1	0	93,285	35.72 MB

Database	Docs #	Index # (Er...	Aler...	R...
DB2	3,406,594	1	0	1
DB3	2,567,472	0	0	2
DB4	2,567,472	0	0	1
DB5	3,406,594	1	0	1
DB6	839,122	1	0	1
DB7	839,122	1	0	1
DB1	779,177	0	0	1

Database	Indexed / sec	Mapped / sec
DB2	0	0
DB3	0	0
DB4	0	0
DB5	0	0
DB6	0	0
DB7	0	0
DB1	0	0

	Database	Size
🟦	DB1	772.33 MB
🟨	DB2	4.63 GB
🟩	DB3	4.25 GB
🟥	DB4	4.25 GB
🟪	DB5	4.63 GB
🟦	DB6	1.63 GB
🟥	DB7	1.63 GB

Figure 6. Example of the basic RavenDB User Interface and the control panel.

Timeframe for implementing, documenting and testing the sample database is also limited, so quick-to-learn and implement technology took priority.

The next section of this study will also act as a brief documentation on how to get started with RavenDB. This section will only cover the steps to downloading the database, creating a simple example database and then populating it with sample documents. This study will not go into more advanced features due to time restraints.

4.1.1 Setting up the database

The first step is to download RavenDB from the official distribution site (RavenDB, n.d.). Figure 7 shows the home screen of RavenDB. The test database will be built on a Windows environment, so the process for different operating systems might vary.

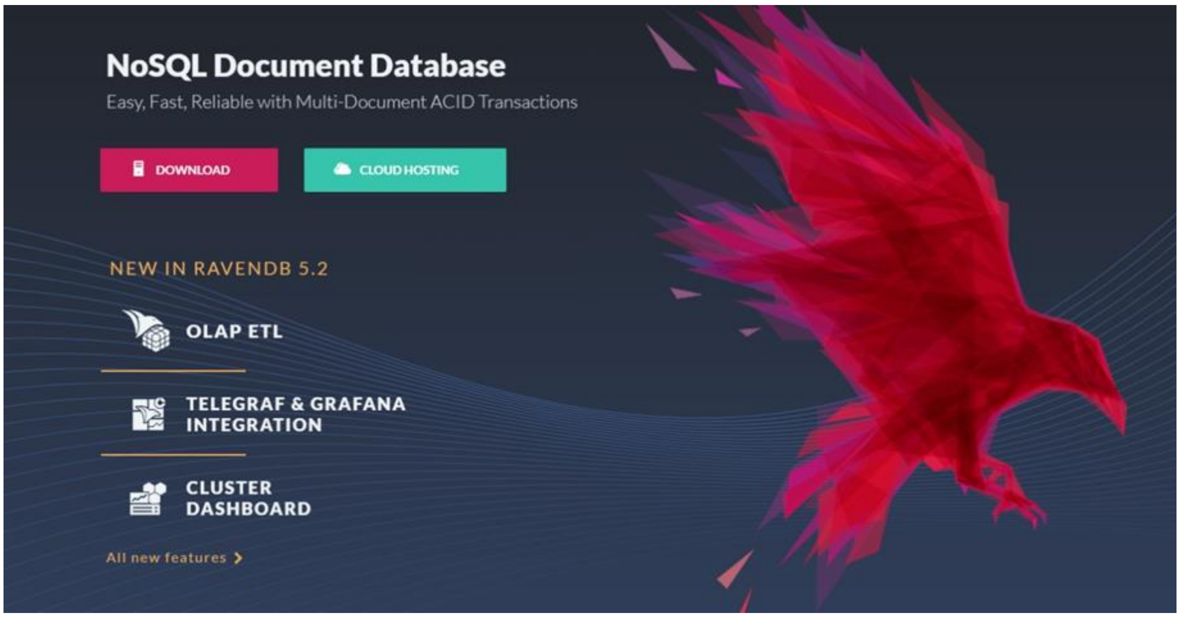
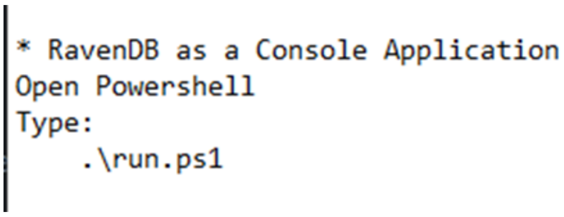


Figure 7. RavenDB download page and the location of the download button.

After downloading the package, the installation process can be started via the Powershell command:



```
* RavenDB as a Console Application
Open Powershell
Type:
    .\run.ps1
```

Alternatively, the installation can be started by extracting the newly downloaded package. The extraction process can be started by right-clicking the .zip file.

Figure 8 will illustrate the location of the extraction action.

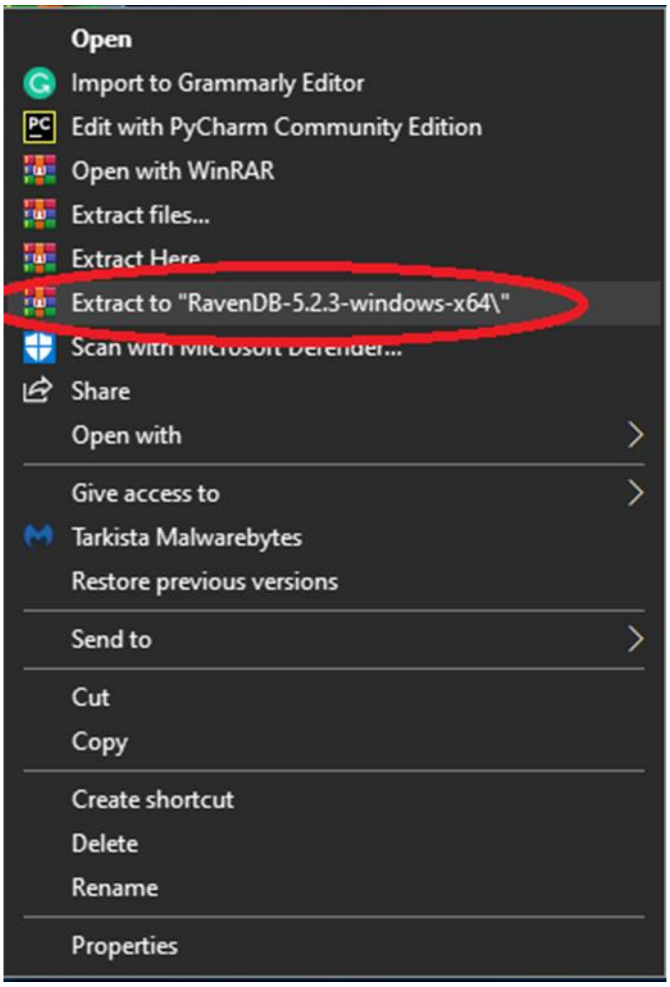

Open
Import to Grammarly Editor
Edit with PyCharm Community Edition
Open with WinRAR
Extract files...
Extract Here
Extract to "RavenDB-5.2.3-windows-x64\"
Scan with Microsoft Defender...
Share
Open with
Give access to
Tarkista Malwarebytes
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

Figure 8. The location of the Extract button.

Once the extraction process is finished, the installation can be started by navigating to the newly created folder and locating the file called “run.ps1”. Installation is started by right-clicking the file and choosing the “Run with PowerShell” option. After running the command, the RavenBD Setup Wizard prompts the user to choose the security settings for the new database. For the purpose of this test, the unsecured free database setup will be used, as the database will only be used inside of a controlled testing environment.

After successful setup, the user will be directed to the RavenDB control panel screen. Figure 9 illustrates how this screen looks.

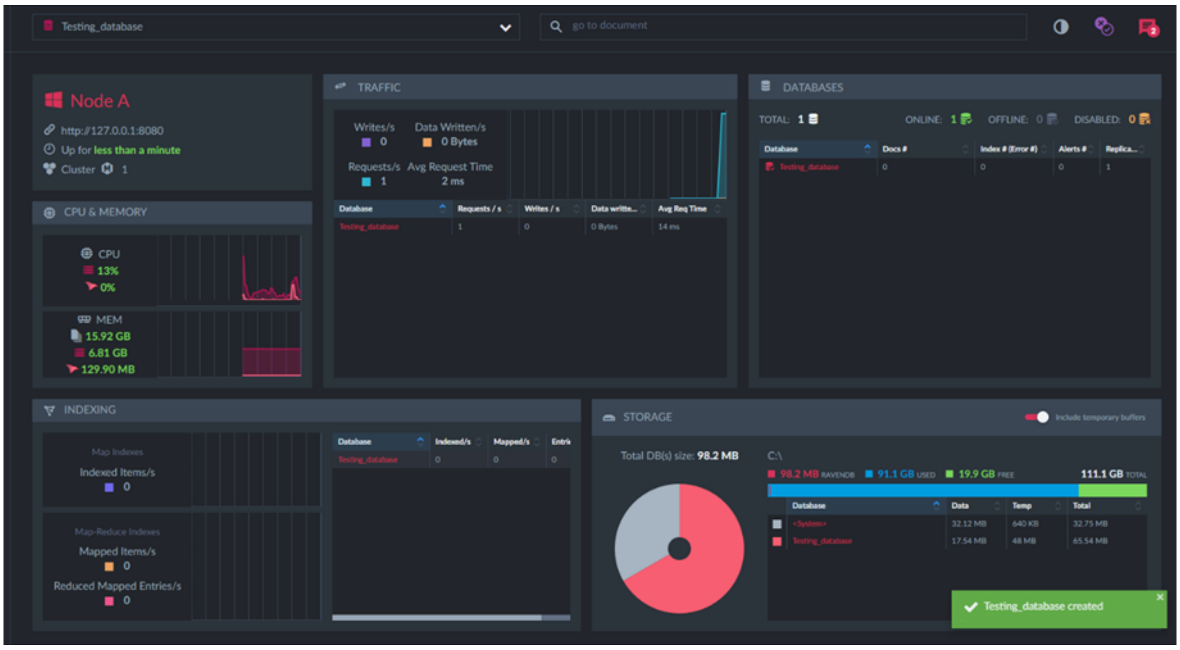


Figure 9. RavenDB Control panel.

RavenDB database is now up and running. RavenDB provides a pre-built feature that allows the users to create a pool of sample documents to populate the new database with. This feature can be found in the sidebar. See Figure 10.

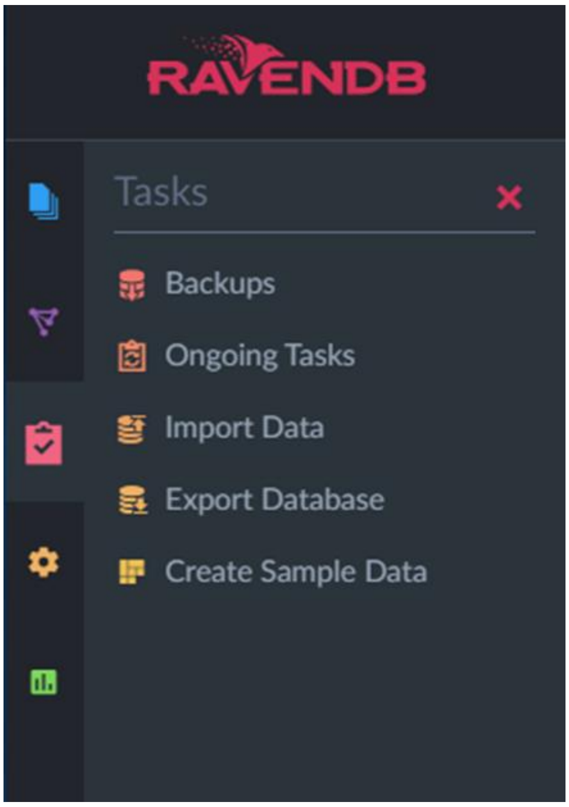
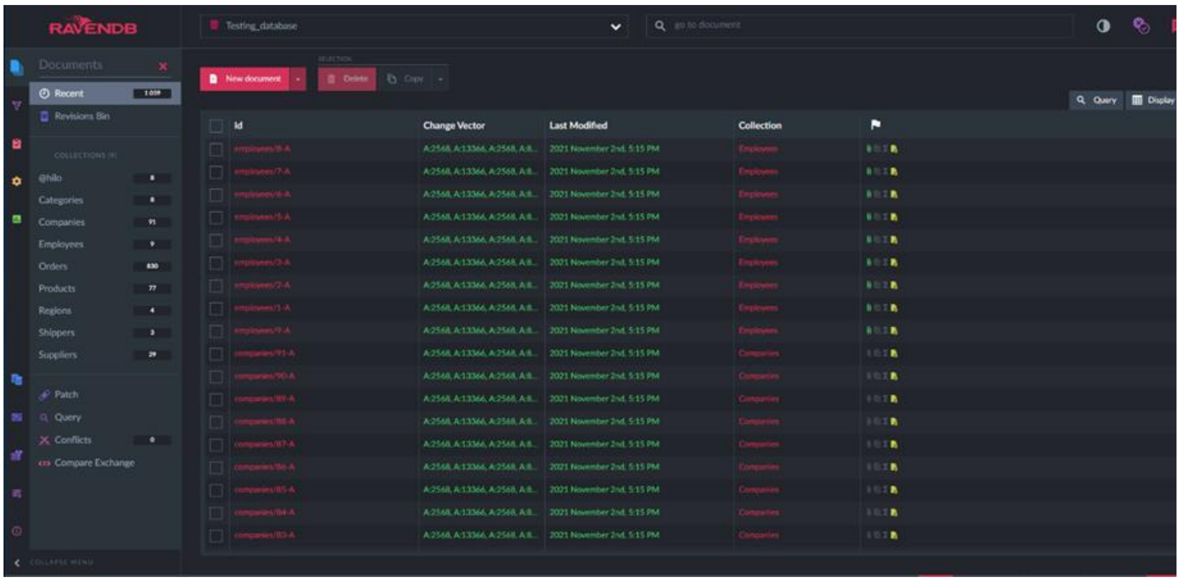


Figure 10. RavenDB Taskbar.

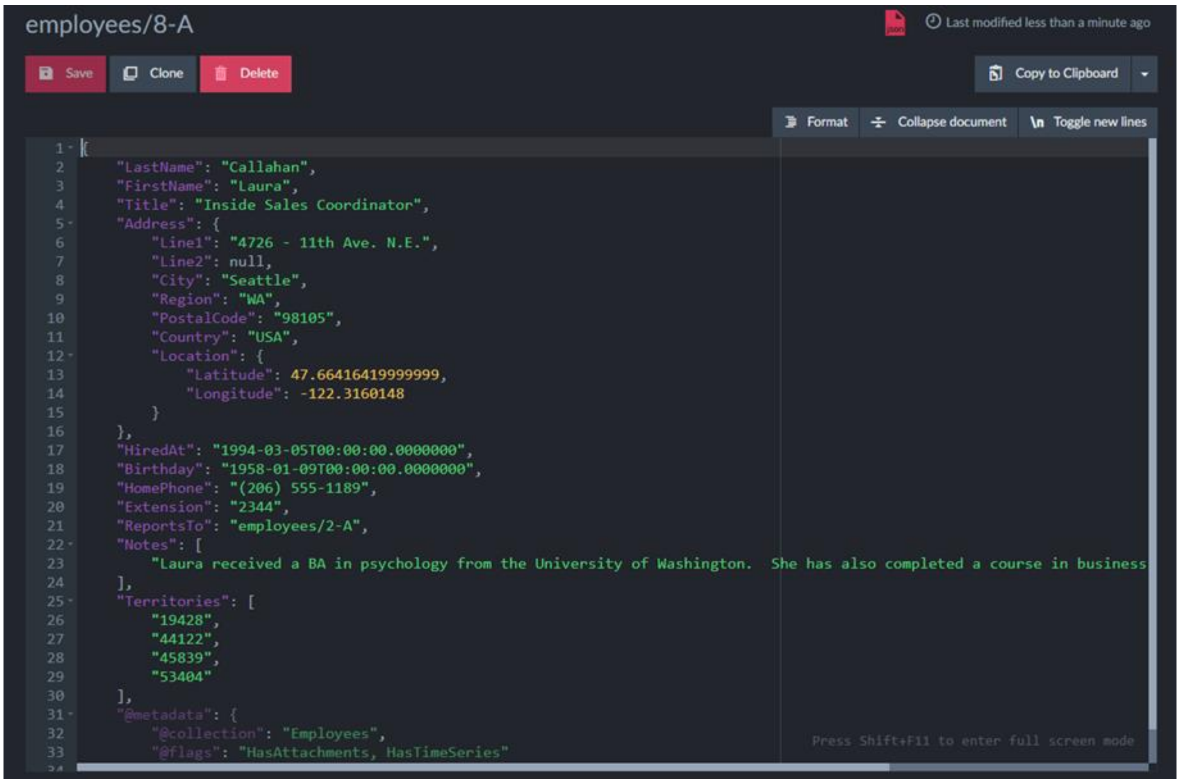
By selecting the task “Create sample data” in the sidebar, documents will be created and stored in the documents; see Figure 11.



Id	Change Vector	Last Modified	Collection	
employees/8-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/7-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/6-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/5-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/4-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/3-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/2-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/1-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
employees/9-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Employees	
companies/91-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/90-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/89-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/88-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/87-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/86-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/85-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/84-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	
companies/83-A	A:2568, A:13366, A:2568, A:8...	2021 November 2nd, 5:15 PM	Companies	

Figure 11. The newly created database will be populated with sample documents.

All data is not generated and stored as JSON data, which the user can now populate the practise database with.

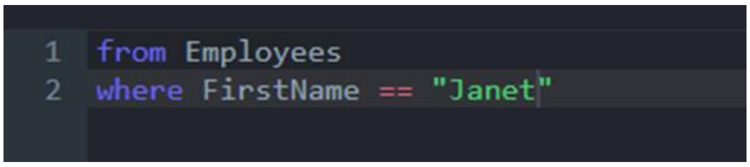


```
{
    "LastName": "Callahan",
    "FirstName": "Laura",
    "Title": "Inside Sales Coordinator",
    "Address": {
        "Line1": "4726 - 11th Ave. N.E.",
        "Line2": null,
        "City": "Seattle",
        "Region": "WA",
        "PostalCode": "98105",
        "Country": "USA",
        "Location": {
            "Latitude": 47.66416419999999,
            "Longitude": -122.3160148
        }
    },
    "HiredAt": "1994-03-05T00:00:00.0000000",
    "Birthday": "1958-01-09T00:00:00.0000000",
    "HomePhone": "(206) 555-1189",
    "Extension": "2344",
    "ReportsTo": "employees/2-A",
    "Notes": [
        "Laura received a BA in psychology from the University of Washington.  She has also completed a course in business
    ],
    "Territories": [
        "19428",
        "44122",
        "45839",
        "53404"
    ],
    "@metadata": {
        "@collection": "Employees",
        "@flags": "HasAttachments, HasTimeSeries"
```

Figure 12. All RavenDB data is stored as JSON files.

4.1.2 Running simple queries

RavenDB uses a query language called RQL (Raven Query Language), which is designed to be similar to SQL query languages. This makes it easier for the end-users to shift from using SQL based queries to RQL queries.



```
from Employees
where FirstName == "Janet"
```

Figure 13. A basic query from the 'Employees' database, where 'FirstName' equals Janet.

4.1.3 Conclusion

From the tests performed with RavenDB, the following conclusions could be made: Raven DB was found slightly inconvenient to get started with, even though the RQL querying language is rather similar to the SQL querying languages. In the free trial, populating the database with custom documents was seen to be too time-consuming, so the document package offered by RavenBD was used instead.

However, once the basic queries had been learned, pulling data from the database was seen to be efficient and easy.

The UI and the control panel was clear and easy to use. However, this might be due to the missing features in the free trial client.

Performance-wise, RavenDB was running smoothly and without any issues during the whole testing period.

4.2 MongoDB

For this database performance test, MongoDB Atlas will be used, as it is the most accessible option for non-subscription users and offers improved service and security features for an enterprise-level subscription. MongoDB Atlas supports Windows, macOS and Linux operating systems.

4.2.1 Setting up the database

On the MongoDB homepage (MongoDB Atlas, n.d.) the option 'start free' is chosen. As a next step, an account needs to be created. In the overview of products which follows the option ‘Visit MongoDB Atlas’ needs to be selected. This leads to the ‘Create your first database’ starting page; see figure below.

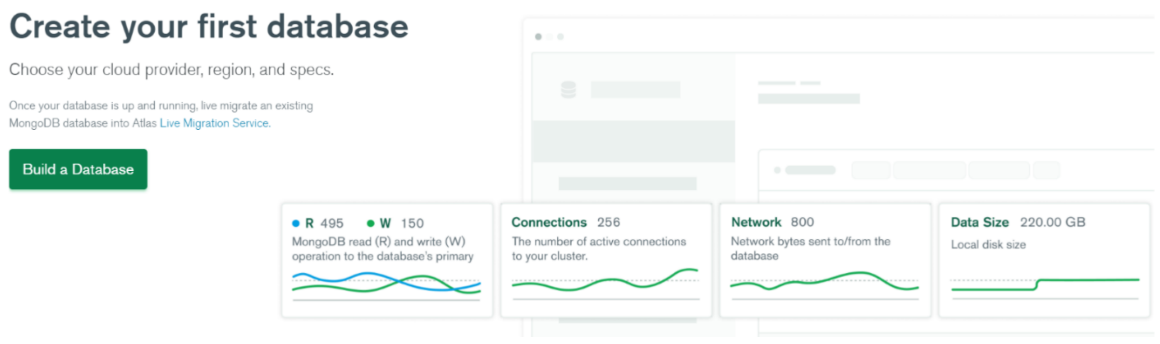
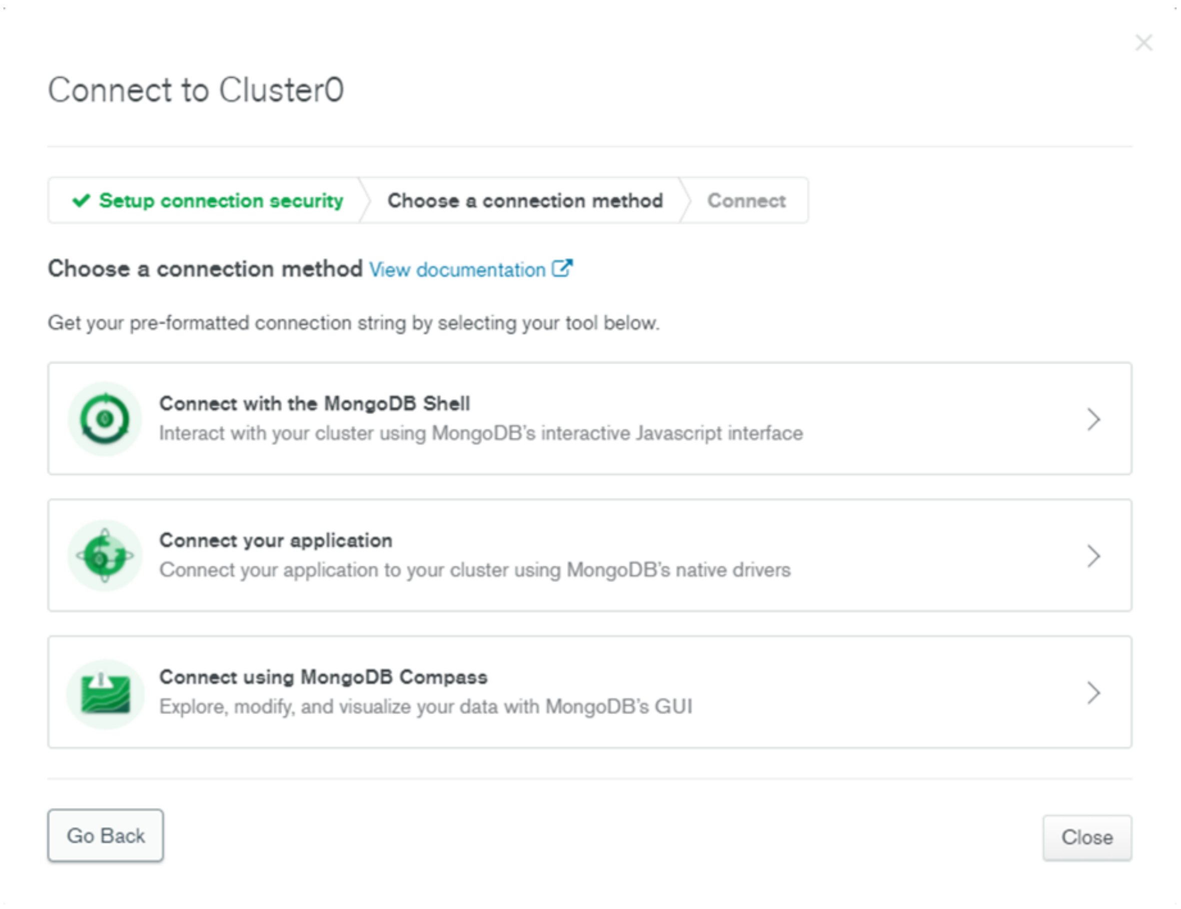

Create your first database
Choose your cloud provider, region, and specs.
Once your database is up and running, live migrate an existing MongoDB database into Atlas Live Migration Service.
Build a Database
● R 495 ● W 150
MongoDB read (R) and write (W) operation to the database's primary
Connections 256
The number of active connections to your cluster.
Network 800
Network bytes sent to/from the database
Data Size 220.00 GB
Local disk size

Figure 14 MongoDB starting screen for creating a database.

In the next screen, the option to create a shared cloud database is selected since this is the only one free of charge. After selecting a cloud provider and a region a so-called cluster can be created. The next screen is the Security Quickstart.

After creating the MongoDB Atlas account and choosing the cluster, connecting to MongoDB Shell requires some additional steps. “Connect with the MongoDB Shell” will start the process. In the next screen, the user will be prompted to download and install mongosh 5.1

After the download, a simple installation is performed via the installation wizard. After that, the user has to add mongosh to their PATH variable.


Connect to Cluster0

✓ Setup connection security | Choose a connection method | Connect

Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

Connect with the MongoDB Shell
Interact with your cluster using MongoDB's interactive Javascript interface

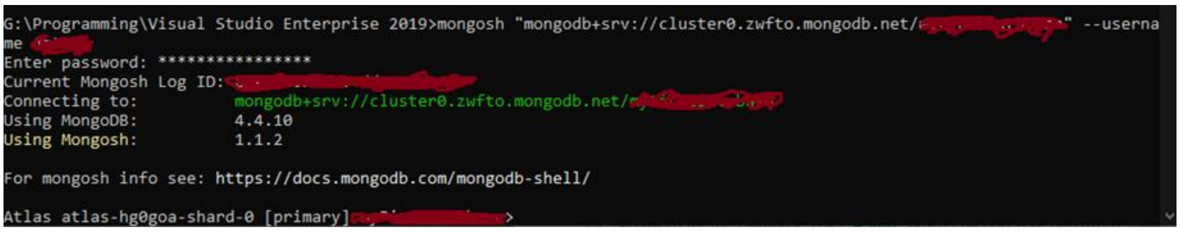
Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI

Figure 15. Cluster selection view.

Once mongosh has been successfully installed and configured, the MongoDB Atlas control panel will give the user their unique connection string that can be used to access the data-base via shell command.

mongosh "mongodb+srv://cluster0.zwfto.mongodb.net/{database name}" --username {user name}


```
G:\Programming\Visual Studio Enterprise 2019>mongosh "mongodb+srv://cluster0.zwfto.mongodb.net/[redacted]" --userna
me [redacted]
Enter password: ****************
Current Mongosh Log ID:[redacted]
Connecting to:          mongodb+srv://cluster0.zwfto.mongodb.net/[redacted]
Using MongoDB:          4.4.10
Using Mongosh:          1.1.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-hg0goa-shard-0 [primary][redacted]>
```

Figure 16. Successful connection (with identifying info covered)

A new database can be either created from the dashboard or the shell. The shell command for creating a new basic database is:

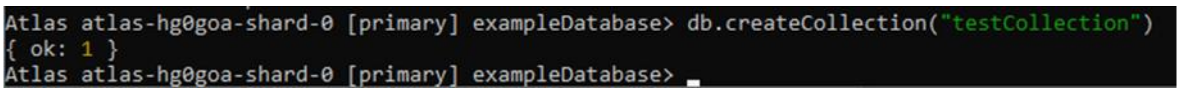
use DATABASE_NAME

4.2.2 Queries

After creating the database, it is now possible to populate it with sample data. To create an empty collection, the user inputs the following query:

db.createCollection("CollectionName")

Figure 17 illustrates how the successfully performed query looks like when the collection is created.



```
Atlas atlas-hg0goa-shard-0 [primary] exampleDatabase> db.createCollection("testCollection")
{ ok: 1 }
Atlas atlas-hg0goa-shard-0 [primary] exampleDatabase> _
```

Figure 17. Collection created successfully.

Next, a simple JSON data document is created, see Figure 18 for the example syntax.

```
> db.testCollection.insertOne(
        {
                First_Name: "Mary",
                Last_Name: "Sue",
                Date_Of_Birth: "1992-07-18",
                e_mail: "mary_sue.123@gmail.com",
                phone: "02343784"
        })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

Figure 18. A simple example document for MongoDB database.

Alternatively, the same can be achieved via the MongoDB Atlas control panel.

Now the data query can be performed to confirm the data has been created and put into the 'testCollection' collection as it should. If the newly created JSON document has been inserted into the collection successfully, the following query illustrated in Figure 18 should return data instead of an empty field. The user can query the whole collection by running the command:

>db.COLLECTION_NAME.find()

If a more precise query needs to be made, filter words can be used:

```
db.COLLECTION_NAME.findOne({First_Name: "Mary"})
```

Figure 19. Example query with filters for MongoDB.

This should return all the fields where the First_Name value is “Mary”

4.2.3 Conclusion

MongoDB was slightly more difficult to get started with than RavenDB due to their different approach to the shell connection. The need to install a mongosh added an extra step to the setup process, but it was noted the connection via mongosh was more stable and easy to use after the installation was done. This also makes future connections to the system a bit easier, since the connection is always done via an unique mongosh credential key that automatically connects the user to the database.

Command prompt queries work similarly to RavenDB’s PowerShell queries. However, compared to the RavenDB control panel, the MongoDB Atlas control panel is not quite as user friendly and clear. Another disadvantage of MongoDB Atlas was that it was also very difficult to add documents to the database from the control panel itself. The documents would not often show up correctly or they were incorrect or messy. This might get easier with additional training but during the performed tests, RavenDB’s control panel felt more accessible.

4.3 MongoDB vs RavenDB

4.3.1 Usability

In these practical tests usability aspects of both databases have been examined. Indicators for this usability are:

- The ease with which the database could be set up,
- The clarity of the user interface and the control panel
- The accessibility of the documentation.

Setup for RavenDB proved to be slightly troublesome and in the testing phase, some problems with PowerShell capability were encountered. This required using some outside resources. Even though, setting up RavenDB was easier and faster when compared to MongoDB.

The user interface and the control panel for both of the tested databases were clear and easy to use, with RavenDB being slightly more accessible. This can be due to limits in the free trial version since the free trial of MongoDB Atlas offered a slightly more diverse selection of settings and features.

Running queries on both tested databases was simple and the learning curve is not too intimidating. This especially holds for if the end-user has previous experience with SQL languages since both options are specifically designed to function in a similar syntax. However, RQL, the RavenDB syntax is more similar to SQL syntax, so transitioning from SQL to RQL would be significantly easier.

Both technologies have good, well accessible documentation that covers both basic and more advanced level queries and during the testing phase. There was no issue with running sample queries even with little or no experience with NoSQL query languages.

During the tests, queries made with the RavenDB test database proved to be slightly faster and more performance efficient. Again, this might be due to the limitations the free trial of RavenDB has in features since the stripped-down version is most likely to be more light-weight.

Both databases offer multiple levels of data protection via their tiered subscription model. However, more research is needed due to MongoDB having had some issues with data security in the past (Spadafora, A. 2020).

4.3.2 Performance

Both RavenDB and MongoDB ranked well in the performance test, with the data being available without issues and easily. Both options offer real-time access with their cloud-based control panel and both RavenDB and MongoDB can also be accessed via quick and fast-performing shell connection.

With both RavenDB and MongoDB to access the free trial version of the database, continuous access to the internet is needed. This is due to how the connection to the database is handled via the control panel environment. However, both RavenDB and MongoDB subscription services offer ways to securely access the data even with no internet connection. This is worth noting when conducting further research on the topic since the Dutch tax office must be able to access the data in cases where there is no internet connection.

Updating data is easy for both database solutions and their documentation offers good guidance on the different operations and functions available.

Since both RavenDB and MongoDB Atlas use cloud-based control panels that allow easy and quick connection from multiple different authorized instances, accessing the database from multiple data centres is not an issue. Security-wise, RavenDB's cloud access uses two-factor authentication to make multiple connections more secure, in addition to the control panel management, where new access rights can be granted and revoked. MongoDB also offers a way to control user credential access from the control panel, but it is unclear if two-factor authentication is available. Since these features are not available for free trial users, so it is recommended that further research looks more into the available options within the subscription tiers.

Both RavenDB and MongoDB offer standard backup management features in case of data corruption, however, it is important to notice that the backups have to be manually managed and set up during the initial database setup and backing up large amounts of data requires additional investments on top of the initial subscription costs. Backups are also not available for free users.

4.3.3 Flexibility

RavenDB promises high flexibility and future technology access coverage. This means, that if the database needs to expand, accommodate new types of data and adapt to future technologies, RavenDB should stay updated and maintained by the vendor. This is especially important for the Dutch tax office since the need to store different types of data might change over time. MongoDB promises similar features, in addition to that updating the cluster can be done with no downtime in the system, meaning the database will stay available even during the update.

Both systems are highly flexible and elastic, offering good possibilities for future expansion, for example in terms of how much and what kind of data can be stored.

5 Conclusions and recommendations for further research

5.1 Conclusions

The objective of this study is to advise the Dutch Tax office which type of NoSQL database best suits their needs. To fulfil this objection, the question *“How well do the different types of databases perform on the key performance indicators”* needs to be answered. The NoSQL databases included in these studies were: MongoDB (document store), Redis (key-value store), Cassandra (wide column store), Neo4j (graph database) and RavenDB (document store). As described in section 2.2, key performance indicators used in these studies are usability, performance and flexibility.

5.1.1 Conclusions on the usability of the examined databases

To estimate the usability of a specific database the following questions have been formulated to determine how well the chosen database technologies fulfil the needs of the Dutch tax office:

1. Is it possible to store all information types needed?
2. Can it handle the type of queries that are needed for accountability?
3. Can the data be protected?
4. How much effort is storing data and making queries?

Based on both the theoretical framework as presented in section 3 and the practical comparison between MongoDB and RavenDB the following conclusions can be made:

- For a graph database like Neo4j, it is not clear whether or not they can handle both structured and unstructured information. Since it has a restricted application domain graph databases will most probably not be suitable for the Dutch Tax office. The other three types of databases can store all information types needed. However, some authors have mentioned that Redis is mostly useful for situations in which simple data models can be used and applications have frequent short readings
- These databases can store different information types does not automatically mean that they can also be included in a combined query, for example, *“find all information about allowances for childcare in the last 5 years*”. The literature examined for this study did not give much insight into this aspect. An exception is Cassandra from which it is known that the search condition is limited to a fixed key value or a range of key values. It was out of scope for the practical comparison.

- Both MongoDB and RavenDB offer multiple levels of data protection. However, it was discovered that there have been security issues with MongoDB in the past. No information was found on how well security works for the other databases.
- Since MongoDB doesn't use SQL like syntax for query operations more time has to be invested in learning to work with it than needed for using RavenDB. However, as concluded from the practical tests, they both have good, well accessible documentation.

5.1.2 Conclusions on the performance of the examined databases

For this study, performance is defined by the following questions:

1. Is information continuously available?
2. Can information be updated regularly?
3. Can multiple data centres work with the same database?
4. Is the data protected against corruption?

Literature on Cassandra indicated that one of its advantages is the continuous availability of the stored data. The practical compassion showed that both RavenDB and MongoDB offer cloud-based services, which come with real-time access. In the theoretical study part, no information was found on this aspect for the other types of databases.

As indicated in section 3, document stores are especially suitable for working with quick-changing data and allow for regularly adding new data groups. This was confirmed in the practical tests. For the other databases, no specific mention of this aspect was found.

Both RavenDB and MongoDB Atlas allow easy and quick connection from multiple different authorized instances. Therefore, accessing the database from multiple data centres should not be an issue. In reviews of Cassandra, it was found that they can be operated from multiple data centres with little or no data loss.

For both Cassandra and Redis it was mentioned that there is no mechanism for ensuring integrity. RavenDB and MongoDB Atlas offer standard backup management features in case of data corruption. However, these backups have to be manually managed and backing up large amounts of data requires additional investments.

5.1.3 Conclusions on the flexibility of the examined databases

The Dutch Tax office needs a database that is flexible which means that:

1. The database can easily be expanded (size, type of information)
2. New types of queries can be incorporated
3. It will be possible to make use of newly developed underlying technology

It was concluded that all four types of NoSQL databases can be easily expanded, but some restrictions have to be taken into account. MongoDB allows a limited number of nestings (100 levels) and doesn't support the joining of documents. When using densely linked data, both Redis and Cassandra show a huge decrease in efficiency. They also have the problem that larger numbers of collections lead to more complex structures. Ne4j can only be expanded within the application domain (connected data). Both MongoDB and RavenDB offer good possibilities for future expansion and new data type storage.

Whether or not new types of queries can be incorporated is not clear from the theoretical part of these studies. Except that, as mentioned before, reviews of Cassandra indicate that search conditions are limited to a fixed key value or a range of key values. These reviews also mention that problems can arise with querying on a large amount of data.

5.1.4 Overall conclusions on which type of database fits the needs of the Tax office best

As mentioned in section 2, to be able to make a justifiable decision on what database fits their needs the best, the tax authorities also have to consider the technical, financial, organizational, legal and administrative feasibility of a database solution. Unfortunately, based on the results obtained in this study it is not possible to give well-founded and substantiated advice on the technical feasibility of using a specific NoSQL database solution. Reasons are:

- The functional requirements have not yet been established by the Tax office. For example, the tax authorities are still working on mapping out the amount and types of data that must be stored now and in the near future.
- The situation that has to be solved is extremely complex and no studies are available on similar situations as the Tax office faces.
- It was not possible to work with real data from the Tax office.
- A practical test could only be performed with free trial versions.

Therefore, the original aim of this study cannot be achieved. However, a lot of insight was gained on necessary topics for further research. In addition, the following, more general conclusions can be made regarding the use of NoSQL databases:

- It is most likely that using a NoSQL database is a feasible option for the Dutch tax office
- Graph databases don't fit the functional demands of the Tax office. A key store database also doesn't seem acceptable. Of the four types of NoSQL solutions, a document style database seemed to be the most feasible.

5.2 Recommendations for further research

Several recommendations can be formulated based on the conducted theoretical study on NoSQL databases and the practical tests with MongoDB and RavenDB. These recommendations fall into two categories: recommendations for actions that the tax authorities should execute themselves and recommendations for further research and testing they should invest in.

5.2.1 Recommendation for actions to be taken

As part of the Dutch government, the tax authorities must deal efficiently and effectively with public money. Consequently, investing in a new DMS and potentially a NoSQL database solution should be the solution to a problem that the tax authorities are experiencing or facing in the (near) future. To be able to assess how this problem can best be tackled, it must first be clear what the core problem is. In addition, clear indicators need to be established on what this solution needs to provide for now and in the future. Therefore, the following recommendations can be made:

- Make a thorough estimate of how much information and what kind of information needs to be stored. Including an estimate of how this will change in the future
- Make an overview of the requirements for the management of this information (minimum and maximum retention periods, necessary security measures, etc)
- Make an overview of the type of queries that are expected and the type of information that need to be accessed for these queries.
- Decide on functional demands and divide these between 'need to have' and 'nice to have'.

- Set SMART performance indicators

5.2.2 Recommendation for further research

To be able to decide if NoSQL databases are suitable for the Dutch tax office and choose which NoSQL solutions fit the functional demand and performance indicators best, substantially more knowledge on and practical experience with NoSQL databases is needed. To obtain the necessary information the following recommendations are given:

- Have a more extensive and in-depth study carried out on the characteristics of the four types of NoSQL databases. In addition, search for comparable organizations who have already decided on using a NoSQL solution (or decided not to use this type of database) and learn from their experience.
- Establish a trend watching program in the field of technical and social developments.
- Perform an extensive test with the most promising database solutions. First with test information and queries specifically created for this purpose. In a second stage with actual data and queries from the tax office.
- Investigate not only the technical feasibility of the different database solutions but also the financial, organizational, legal and administrative feasibility.

References

Apache (n.d.). Open Source NoSQL Database. Retrieved on 10 October 2021. Available at https://cassandra.apache.org/_/index.html

Burleson Consulting (1996). The CODASYL Network Model. Retrieved on 3 March 2021. Available at http://www.remote-dba.net/t_object_codasyl_network.htm

Capterra (n.d.) Apache Cassandra Reviews. Retrieved on 10 October 2021. Available at https://www.capterra.com/p/148406/Apache-Cassandra/reviews/

Codd, E.F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377-387. A Framework for the Evaluation of NoSQL Databases for Big Data Use Cases. Retrieved on 10 October 2021. Available at https://www.igi-global.com/chapter/a-framework-for-the-evaluation-of-nosql-databases-for-big-data-use-cases/256670

Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, Sears, R. (2010). Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM symposium on Cloud computing. Retrieved on 4 November 2021. Available at https://dl.acm.org/doi/10.1145/1807128.1807152

DB Engines (2021 a). DB-Engines Ranking of Relational DBMS. Retrieved on 10 October 2021. Available at https://db-engines.com/en/ranking/relational+dbms

DB Engines (2021 b). DB-Engines Ranking of Key-value Stores. Retrieved on 10 October 2021. Available at DB-Engines Ranking - popularity ranking of key-value stores

DB Engines (2021 c). DB-Engines Ranking of Wide Column Stores. Retrieved on 10 October 2021. Available at DB-Engines Ranking - popularity ranking of wide column stores

DB Engines (2021 d). DB-Engines Ranking of Graph DBMS. Available at https://db-engines.com/en/ranking/document+store

DB Engines (2021 e). DB-Engines Ranking of Document Stores. Retrieved on 10 October 2021. Available at DB-Engines Ranking - popularity ranking of document stores

Endres, J., Bernsteiner, R.C., Ploder, C. (2020).
Flores, A., Ramirez, S. Toasa, R., Vargas, J., Urvina, R., Lavin, J.M. (2018). Performance Evaluation of NoSQL and SQL Queries in Response Time for the E-government. Available at https://ieeexplore.ieee.org/abstract/document/8372362

Foote, K.D. (2018). Available at A Brief History of Non-Relational Databases - DATAVERSITY

IBM (n.d. a). IBM's 100 Icons of Progress. Retrieved on 29 October 2021. Available at https://www.ibm.com/ibm/history/ibm100/us/en/icons/

IBM (n.d. b). Relational Database. Retrieved on 29 October 2021. Available at https://www.ibm.com/ibm/history/ibm100/us/en/icons/reldb/

Kim, S., Kanwar, Y.S. (2019). GeoYCSB: A Benchmark Framework for the Performance and Scalability Evaluation of NoSQL Databases for Geospatial Workloads. Available at https://ieeexplore.ieee.org/abstract/document/9005570

Knowledgenile (n.d. a). Understanding the Pros and Cons of MongoDB. Retrieved on 10 November 2021. Available at https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb/

Knowledgenile (n.d. b). Top 7 Use Cases of Redis. Retrieved on 10 November 2021. Available at https://www.knowledgenile.com/blogs/redis-use-cases/

Kunda, D., Phiri, H. (2017). A Comparative Study of NoSQL and Relational Database. Available at https://ictjournal.icict.org.zm/index.php/zictjournal/article/view/8

Krstić, L.J., Krstić, M.S. (2018). Testing the performance of NoSQL databases via the database benchmark tool. Available at Microsoft Word - 11 15928-86721-1-CE Krstic (ceon.rs)

Lavi, S. (2021). MongoDB Vs RavenDB. Retrieved on 22 November 2021. Available at https://www.itqlick.com/compare/mongodb/ravendb

Lourenço, J.R., Cabral, B., Carreiro, P., Vieira, M., Bernardino, J. (2015) . Choosing the right NoSQL database for the job: a quality attribute evaluation. Journal of Big Data 2 (18). Available at https://doi.org/10.1186/s40537-015-0025-0

Nurhadi, H., Kadir, R.B.A., Surin, E.S.B.M. (2020). Evaluation of NoSQL Databases Features
and Capabilities for Smart City Data Lake Management. Information Science and Applications – Proceedings of ICISA 2020, 383-392. Retrieved on 12 September 2021. Available at https://www.researchgate.net/profile/Nurhadi-Hadi/publication/350586264_Evaluation_of_NoSQL_Databases_Features_and_Capabilities_for_Smart_City_Data_Lake_Management/links/617e961deef53e51e10ddc8c/Evaluation-of-NoSQL-Databases-Features-and-Capabilities-for-Smart-City-Data-Lake-Management.pdf#page=379

Ploetz, A., Kandhare, D., Kadambi, S, Wu, X. (2018). Seven NoSQL databases in a week. Available at
https://books.google.nl/books?hl=nl&lr=&id=irZTDwAAQBAJ&oi=fnd&pg=PP1&dq=fundamentals+NoSQL&ots=rcXbIH3nrH&sig=hZOV5gPtK8sIe6RGJKX_p-yFRkI#v=onepage&q=fundamentals%20NoSQL&f=false

Radoev, M. (2017). A Comparison between Characteristics of NoSQL Databases and Traditional Databases. Computer Science and Information Technology 5(5): 149-153. Available at https://www.researchgate.net/profile/Mitko-Radoev/publication/321799978_A_Comparison_between_Characteristics_of_NoSQL_Databases_and_Traditional_Databases/links/5e82dc62299bf1a91b8d1a37/A-Comparison-between-Characteristics-of-NoSQL-Databases-and-Traditional-Databases.pdf

RavenDB (2019). RavenDB vs MongoDB: Performance, Cost, and Complexity. Retrieved on 22 November 2021. Available at https://ravendb.net/articles/ravendb-vs-mongodb-performance-cost-and-complexity

RavenDB (n.d.). NoSQL Document Database. Retrieved on 10 October 2021. Available at https://ravendb.net/

Redis (n.d.). Redis. Retrieved on 12 October 2021. Available at https://redis.io/

RubyGarage (n.d.). Capabilities of the Neo4j Graph Database with Real-life Examples. Retrieved on 12 October 2021. Available at https://rubygarage.org/blog/neo4j-database-guide-with-use-cases

Spadafora, A. 2020. Millions of online shoppers have data exposed. Techradar. Retrieved on 24 November 2021. Available at https://www.techradar.com/news/amazon-and-ebay-shoppers-data-exposed-online

Strozzi, C. (2007). Why NoSQL, in the first place? Retrieved on 18 August 2021. Available at http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Philosophy%20of%20NoSQL

TrustRadius (n.d.) Cassandra reviews. Retrieved on 12 October 2021. Available at https://www.trustradius.com/products/apache-cassandra/reviews

Warda, S. (2019). MongoDB vd RavenDB. Retrieved on 22 November 2021. Available at https://indexoutofrange.com/RavenDBvsMongoDB/

Weekly Webtips (2020). Redis — What and Why?. Retrieved on 12 October 2021. Available at https://medium.com/weekly-webtips/redis-what-and-why-pros-cons-ae2f5bc750fd

Wikipedia (n.d.). Relational database. Available at https://en.wikipedia.org/wiki/Relational_database#Relationships

Zollmann, J. (2012). NoSQL Databases. Available at https://www.csee.umbc.edu/courses/graduate/691/fall18/data-science/nosql_chapter.pdf#cite.EdlichFriedlandHampeBrauer201010

Appendix A Glossary

Availability: what percentage of time a system is operating correctly

CODASYL: Conference On Data System Language (network database model developed in the 1960s)

Consistency: The reliability of the performance of the functions of a database

DBMS: Database Management System

DMS: Document Management System.

Durability: The requirement that data be valid and committed to disk after a successful transaction

Elasticity:

GDPR: General Data Protection Regulation (Regulation which lays down rules relating to the protection of natural persons concerning the processing of personal data and rules relating to the free movement of personal data).

IMS: Information Management System (hierarchical database model developed by IBM in the 1960s)

Labels: Used to group nodes in a graph database. Each node can be assigned multiple labels. Labels are indexed to speed up finding nodes in a graph.

Latency:

Maintainability: The ease with which a product can be maintained, i.e., upgraded, repaired, debugged and met with new requirements.

Nodes: The main data elements in graph databases that are interconnected through relationships. A node can have one or more labels (that describe its role) and properties (i.e. attributes).

NoSQL: Not-only sequel

Relationships: Connects two nodes in a graph database. Nodes can have multiple relationships. Relationships can have one or more properties.

TCI: Total implementation cost

TCO: Total cost of ownership

YCSB: Yahoo! Cloud Serving Benchmark (YCSB), proposed and implemented by Cooper et al. (2010)