INFO - H - 415
2017 - 2018

18 December 2017

Document stores and MongoDB

Project: Advanced Databases

Kaïs Albichari	000395807	M-INFOS
Tanguy d'Hose	000409718	M-INFOS

Table of contents

1. Introduction to the scenario

1.1. Company activity and needs

The startup requiring database consultancy is *Nahmazon*; a small online book store. The company will start its commercial activities by selling books, but if its revenue is decent enough, it will expand to related products and beyond.

At the moment, the startup would like to have databases that are easily manageable, understandable and not too constraining. If tradeoffs exist, the company would like to focus on ease of management and understanding.

In case the startup becomes successful, there will be need for expansion. As a result, the chosen DBSA should be able to scale, admit new inventory and offer data security by using cloud technologies or multiple backups.

1.2. Currently implemented solution

As the *Nahmazon* team did not dispose of sufficient time to analyze several solutions, the startup went for the most popular solution: relational databases. Currently, the databases are managed using SQL languages.

1.3. Future infrastructure requirements

As mentioned previously, the founders of Nahmazon hope to have massive sales and become successful. In order to prevent future problems, they would like to have an infrastructure that easily scales, admits new inventory and offers data security.

All in all, Nahmazon would like to know whether the choice of relational databases and SQL languages is the most adapted to the company’s goals and, if not, what would be a more suiting alternative.

2. NoSQL Databases

2.1. Introduction to NoSQL databases

2.1.1. Definition

NoSQL refers to a wide variety of models that don’t fit into the relational model. The problems induced by Big Data have led to new ways of data storage and management that don’t correctly fit into the well-known relational approach. That’s why this approach has been named « NoSQL ».

The term NoREL (no relational) is an alternative name, as the only common aspect of all NoSQL databases is their non-relational structure. NoSQL could have been named according to this logic, thus meaning SQL is not required in order to manage NoSQL databases. Instead, other query languages are used. NoSQL could also stand for “Not only SQL” meaning that the systems use SQL along with other technologies and query languages.

2.1.2. Main differences between SQL and NoSQL databases

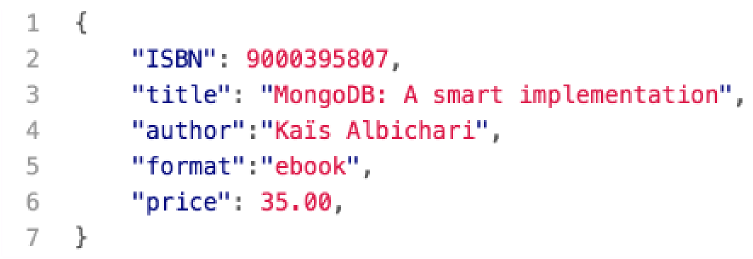
2.1.2.1. Tables

SQL databases provide a store of related data tables. These are rigid; they have to be modelled and created before any data can be manipulated. Once this model is initialized, no major changes can be operated in the model.

ISBN	Title	Author	Format	Price
9000395807	MongoDB: A smart implementation	Kaïs Albichari	ebook	35.00
0000409718	Document Stores: small introduction	Tanguy d’Hose	ebook	40.00

Figure 1: book information structure

Every row of figure 1 represents a book record. In SQL, this structure is static. The table can’t accept the storage or insertion of a wrong type of data. NoSQL databases store their information in a JSON-like field-value pair document.



```
{
    "ISBN": 9000395807,
    "title": "MongoDB: A smart implementation",
    "author":"Kaïs Albichari",
    "format":"ebook",
    "price": 35.00,
}
```

Figure 2: JSON-like field-value pair document

Similar documents are stored in a collection, which is equivalent to an SQL table. The NoSQL DBMSs are more flexible: they accept data without any restriction on the structure. For instance, new attributes could be added when creating a new document for an existing object. These loosened restrictions make NoSQL databases very useful when using agile development techniques.

One the one hand, SQL tables have a strict data model that prevents many mistakes from happening. On the other hand, NoSQL loosened restrictions allow much more actions, but the ability to store any type of data anywhere at any time could possibly cause issues in the long run.

2.1.2.2. Schema

SQL databases require the definition of tables and field types before allowing data manipulation. These are referred to as the schema. The schema can contain additional information, such as primary keys, indexes, relationships and other functionalities such as triggers and stored procedures.

The definition of the schema is close to the concept of static tables. The schema must be designed before any manipulation on the data can occur. If, for some reason, changes must be applied later, their execution could become complicated.

On the contrary, NoSQL databases have no need for a schema. As much information as required can be added without having to create a new collection for it. For instance, the insertion of a document in a “book” collection can be executed without having created the collection on beforehand. The execution of the insert operation will create the collection implicitly and proceed to inserting the document within this new collection.

2.1.2.3. Scalability

NoSQL databases are perfectly suited for horizontal scaling. This means that the data can be split across multiple computers that do not need to know the partitions of data contained on the other computers in order to perform their tasks. In other words, the database is shared among a pool of servers. The main advantage of this approach is that the architecture can be adapted depending on the load. This adjustment is achieved by building a clustered environment, which is a way to perform tasks much more efficiently and independently.

Even though relational DBMSs could also scale horizontally, clustering will make data management a lot harder. SQL DBMSs will rather « scale up », or scale vertically, meaning that resources are added to the single machine/server in order to increase its performances and be able to support the load.

2.1.2.4. Normalization

Minimizing redundancy can be achieved in SQL databases by applying normalization. If a “publisher” attribute was to be added to a “book” record, it wouldn’t be optimal to repeat all information about a publisher in every concerned book record. In order to avoid this from happening, a new table storing publisher’s information should be created. This table could be related to the book table by sharing information, the publisher’s ID for instance. If the publisher has to be updated, this can now be done without having to update information in several book records.

The same principle can be applied to NoSQL databases. Although, this process isn’t always practical. This is the main reason why denormalization is usually applied: all “book” records will contain all information concerning their related “publisher” data object. Since all required information is nested within a same file, it obviously leads to a faster querying speed. When updating a publisher’s information, all related books will also be updated, resulting in a significantly slower update speed.

The use of denormalization in NoSQL databases is due to the lack of a JOIN operator. In case books were have to be joined with publisher’s information, all book documents and all publisher documents would have to be fetched and then all should be linked manually. It goes without saying that this operation is quite time-consuming simply to achieve a join.

2.1.2.5. Data integrity

The capability of enforcing the data integrity using foreign key constraints is a direct consequence of the schema structure in SQL. Foreign key constraints ensure the validity of the key of all records possessing them, thus guaranteeing an entry in the correct table.

Unfortunately, these also introduce new difficulties: removing an entry is impossible while an object is still using a key referring to it. The use of constraints prevents developers or users to add, edit or remove records that if added, edited or removed would lead to invalid data.

In NoSQL databases, data integrity options don’t exist. Since a single document is the source of all information regarding a specific data object, data can be added, edited or removed without having to take other documents into account.

2.1.2.6. Transactions

SQL databases implement the concept of “transaction”, allowing database managers to execute multiple updates at the same time. This results in a success or fail for all planned updates. The use of transactions is really useful when it comes to data integrity.

As multiple documents cannot be updated simultaneously, NoSQL databases do not implement the concept of transactions. However, the modification of a single document is atomic, meaning that the update of multiple values will either lead to a success or fail of all updates.

2.1.3. Characteristics

The term “NoSQL” is used to refer to a particular group of DBMSs that share some common characteristics. In some cases, they might not share all of these.

2.1.3.1. Non-relational

Being non-relational means that the relational model, as proposed by E. F. Codd in 1970, does not fit the requirements. This could be related to having mainly semi- or unstructured data. Another reason could be that the system would eventually need to scale to a large number of computers.

2.1.3.2. Open Source

Being open source is not a requirement for a NoSQL DBMS, but most of the NoSQL DBMSs are open source. Moreover, the NoSQL movement tends to assemble multiple organizations to contribute to developing a single solution.

2.1.3.3. Lack of adherence to ACID principles

ACID stands for Atomicity, Consistency, Isolation and Durability. The aim of all NoSQL DBMSs is to provide high availability and scalability across clustered environments, which means that they could lose in durability or consistency, or at least have to find a balance between these properties.

2.1.3.4. No standard query language

There is no standard query language supported by all NoSQL databases. Some of them have their own query languages, while others can support various languages such as JSON, XQuery, etc.

2.2. Types of NoSQL databases

There are 4 families of NoSQL databases. Since this document focuses on document stores, the 4 families will be briefly explained. In addition to document stores, there exists column stores, key-value stores and the graph stores.

2.2.1. Column Stores

Column stores are also referred to as “extensible record sets”, these databases share some similarity with traditional databases in their use of rows, columns and tables. The main difference is that columns are created for each row, instead of being created by the table structure.

A simple way to visualize this is to consider them as key-value collection where each value can either be a simple data or another key-value collection.

2.2.2. Key-Value Stores

Key-value databases are much faster than their equivalent relational database. This is because key-value databases are conceived in a much simpler way. In these databases, the data records use a primary key to uniquely identify the items.

The increased time-efficiency lies in the fact that the database ignores the type of data that is being stored. It is the application side’s job to know how to interpret the data that is being retrieved and stored. Since each operation only requires one hard drive disk access, it allows faster lookups and data saves within the database.

2.2.3. Document Stores

Document stores use the concept of documents instead of tables. These documents are designed to handle semi-structured data, which wouldn’t fit at all in the relational model.

Whilst the database system does not interpret the data in the key-value stores, a document store is aware of the structure of the stored data. A document can contain different pairs of key-value, key arrays and even nested documents. Different documents can also be grouped into a collection. As document stores form one of the larger topics of this document, they will be explained more profoundly later.

2.2.4. Graph Databases

Some databases keep the items in a list, using an index to travel through these items. Graph databases have a graph structure, where each node contains its own information and information about how it is related to other nodes using pointers.

Graph databases are useful when working with a system in which a lot of links exist between different data objects. This kind of data management is most useful when relationships between stored data objects are important. Therefore, the use of famous algorithms such as the shortest path is a must.

Even though graph databases seem to be the perfect solution for several types of systems and architectures, it cannot always guarantee a better performance: when data is frequently updated, the performance of the system will suffer.

2.3. Infrastructure-compatible NoSQL databases

Nahmazon will require its databases to store potentially massive amounts of products, but also user information and additional information such as comments, reviews, lists of producers and so on. As relational databases are too constraining for rapid expansion and secured backups, we will choose amongst NoSQL databases.

Though all NoSQL databases could potentially be infrastructure-compatible, they aren’t necessarily suitable. The main goal of *Nahmazon* is to sell products, these aren’t necessarily related and could potentially be stored in huge amounts and have different characteristics. Column stores and Key-Value stores will require lots of updates. Graph databases will not be efficient as all entities aren’t necessarily related. For that reason, document stores seem to be the most suitable choice for an online sales startup.

3. Document Store

3.1. Introduction to document-oriented databases

Most products that require data storage and manipulation use relational databases. Relational data-schemas are simple and straightforward. In most applications, it makes sense to represent objects as being sets of relationships. Because these relationships between different types of data are specified within the database schema, the database can be queried using a Structured Query Language (SQL). But, the evolution of data environments and programming techniques also required evolution in database types:

- The introduction of cloud computing significantly decreased deployment and storage costs, but required data to be spread across multiple servers easily without any disruption. Using traditional relational databases for complex cloud-based projects requires multiple large tables to be joined together in order to execute a query. Executing distributed joins is a very complex problem when using relational databases.
- The growing popularity of social-based projects, such as social networks, has boosted the need for unstructured data storage. SQL databases are very efficient at storing structured data, but require workarounds and compromises when it comes to querying unstructured data.
- Agile development techniques require databases to change easily and rapidly reflecting changes in demand and requirements. As SQL databases require the structure to be specified in advance using relationships, time consuming ALTER operations would be necessary to adjust the structure to the requirements.

To loosen the restrictions on the database schema, new ways of storing data, such as the previously discussed NoSQL, have been introduced. These new ways allow data to be grouped together more naturally and logically.

Document stores is one of the most popular ways to store data. A document store database uses a document-oriented model in order to store data. Document store databases store each record and its associated data within a single document, meaning that everything related to a database object is encapsulated together. This way of storing data offers multiple advantages:

- Documents are independent units, making performance better and increasing the ease to distribute data across multiple servers.
- Application logic is easier to write. There’s no need for a translation between application objects and SQL; objects can directly be turned into a document.
- Unstructured data can be stored easily, since the documents contain all keys and values required by the application. As a result, the database doesn’t require to know the schema in advance and costly migrations can be avoided.

Document store databases are characterized by powerful indexing features and query engines. These features are aimed at increasing the ease and speed of execution of many different optimized queries.

Typically, documents inserted into document store databases are created using XML or JSON. A document representing a ‘Student’ object could look like this:

```xml
<student>
    <firstname>John</firstname>
    <lastname>Bananas</lastname>
    <courses>
        <course>
            <coursename>Advanced Databases</coursename>
            <courseSSN>INFO-H415</courseSSN>
            <teacher>E.Zimanyi</teacher>
        </course>
        <course>
            <coursename>Database Systems Architecture</coursename>
            <courseSSN>INFO-H417</courseSSN>
            <teacher>S.Vansummeren</teacher>
        </course>
        <course>
            <coursename>XML and Web Technologies</coursename>
            <courseSSN>INFO-H509</courseSSN>
            <teacher>S.Vansummeren</teacher>
        </course>
    </courses>
</student>
```

Figure 3: XML Student object

```json
{
    "_id" : 176485,
    "firstName" : "John",
    "lastName" : "Bananas",
    "courses" : [
        {
            "courseName" : "Advanced Databases",
            "courseSSN" : "INFO-H415",
            "teacher" : "E.Zimanyi"
        }, {
            "courseName" : "Database Systems Architecture",
            "courseSSN" : "INFO-H417",
            "teacher" : "S.Vansummeren"
        }, {
            "courseName" : "XML and Web Technologies",
            "courseSSN" : "INFO-H509",
            "teacher" : "S.Vansummeren"
        }
    ]
}
```

Figure 4: JSON Student object

3.2. Comparison to relational databases

As mentioned previously, document store databases don't require the specification of the database schema, whereas relational databases rely on a rigorously defined database schema. If we would like to create a relational database matching the documents of figure 3 and figure 4, it would look like this:

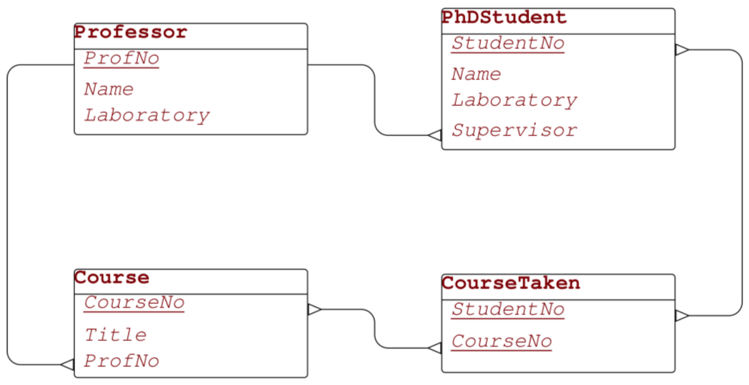


Figure 5: Relational equivalent of Figures 3 and 4[1]

We notice that stand-alone documents require multiple tables and relationships linking them together using primary and foreign keys if they were represented in a relational model.

3.2.1. Main differences between document store and relational databases

3.2.1.1. NoSQL

Document stores use the previously mentioned NoSQL language in order to manage their data. This not the case for all database types; most relational databases use SQL as standard query language.

3.2.1.2. Tables

Relational databases use multiple tables in order to store data. Each one of those tables contains columns, reflecting attributes, and rows representing records. The information about any entity could be contained in one or spread amongst multiple tables. Relationships are used to link associated data from different tables.

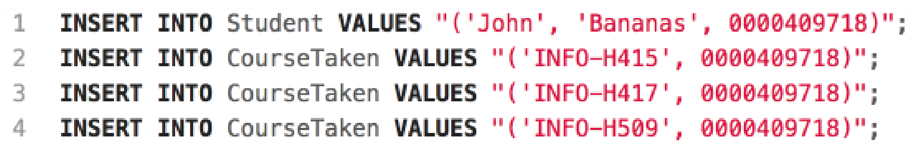
Document databases don't use tables. Instead, they store all data related to a given entity within a single document. As all associated data is stored inside the same document, relationships aren't necessary.

[1] Anonymous. *Session 1 – Active Databases (1/3)* [Exercice notes]. Retrieved from http://cs.ulb.ac.be/public/_media/teaching/infoh415

3.2.1.3. Schemas

When using relational databases, a schema must be provided or created before loading any data into the database. Document stores (and most of NoSQL databases) do not require a previously defined schema in order to function; data can be loaded into the database without having defined any schema.

Having no defined schema means that document stores can take advantage of loosened restrictions when it comes to insertions: two documents could have a different structure and contain different data while still being similar records. For example, inserting a new student in the relational database that would correspond figure 3 would be done this way:



```sql
INSERT INTO Student VALUES "('John', 'Bananas', 0000409718)";
INSERT INTO CourseTaken VALUES "('INFO-H415', 0000409718)";
INSERT INTO CourseTaken VALUES "('INFO-H417', 0000409718)";
INSERT INTO CourseTaken VALUES "('INFO-H509', 0000409718)";
```

Figure 6: insertion of new student in relational model corresponding to figures 3 and 4

If one of the values isn’t specified and not required, the record is still inserted but the missing value is set to null or a specified default value. When two documents differ in this same way, the value that isn’t specified isn’t at all part of the record.

3.2.1.4. Scalability

Document stores easily scale horizontally. They’re perfectly suited for *sharding*: being stored over many thousands of computers whilst maintaining a well-performing system. Relational databases are best suited for vertical scaling. As there exists a limit to the quantity of resources that can be fit inside a machine, horizontal scaling is the best long-term solution and relational databases are considered as being poorly scalable.

3.2.1.5. Relationships

Relational databases use foreign keys in order to enforce relationships between tables. This concept doesn’t exist in document stores. If a link, similar to a relationship in relational databases, exists between documents, the application level would be in charge of handling it.

However, the document store philosophy is to centralize all information related to an object within the same document. Therefore, the need to establish any relationship between documents is less common than in the relational model.

3.3. Document Store DBMSs

As document store databases have gained popularity throughout last years, the number of document store DBMSs have also increased. Here is a non-exhaustive list of these database management systems:

- BaseX
- Caché
- Cloudant
- CouchDB
- CrateIO
- DocumentDB
- ElasticSearch
- HyperDex
- Informix
- Jackrabbit
- MarkLogic
- MongoDB
- OrientDB
- RethinkDB
- SimpleDB
- TokuMX

As MongoDB is by far the most popular document oriented DBMS, we will continue the exploration of document stores trough MongoDB.

4. MongoDB

4.1. Introduction to MongoDB

MongoDB is the most popular document store database. It offers flexibility and scalability. From a development point of view, it is simple for developers to learn and use.

The key features of MongoDB are:

- Stores data in flexible, JSON-like documents
- Data can easily be accessed and analyzed using ad hoc queries, indexing and real-time aggregation
- Distributed database, offering high availability, possibility for horizontal scaling and geographic distribution
- Free and open-source

4.2. Choice motivation

MongoDB is a well-known document oriented DBMS. Throughout the years it has gained in popularity and is increasingly used in applications. The choice of MongoDB enables us to familiarize with newer types of databases, but also to understand why these types of databases are more suited for some software. All in all, we hope this project will give us insight about document oriented databases, and will push us to use MongoDB when developing software that doesn't necessarily need relational databases.

4.3. Install

The installation of MongoDB is rather easy. The installation can be done through Homebrew by following the steps described in MongoDB’s documentation. In this document, the classical way of installing will be described: the manual install.

These are the steps to follow:

1. Download MongoDB’s binary
2. Extract the files from the downloaded archive
3. Copy the extracted file to the target directory
4. Ensure the location of the binaries is in the PATH variable

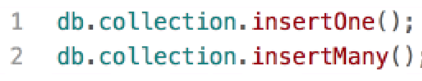
If any difficulties appear when trying to install MongoDB, it is useful to consult their online documentation as it is very complete.

4.4. Main operations (CRUD)

For practical purposes, all operations will be described in Mongo Shell. The syntax of these operations is very similar to the syntax used for combining Mongo with other languages.

4.4.1. Create Operation

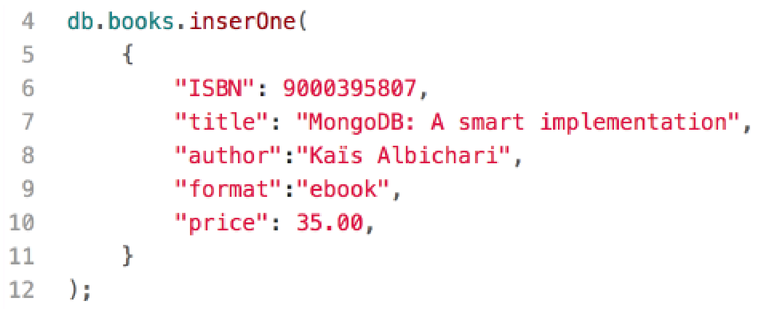
The create operation will add a new document to a collection. As mentioned previously in the document, if the collection doesn’t exist yet, the operation will create it first before adding the new document to it. MongoDB distinguishes two methods to insert documents in a collection:



```
db.collection.insertOne();
db.collection.insertMany();
```

Figure 7: MongoDB’s insertion commands

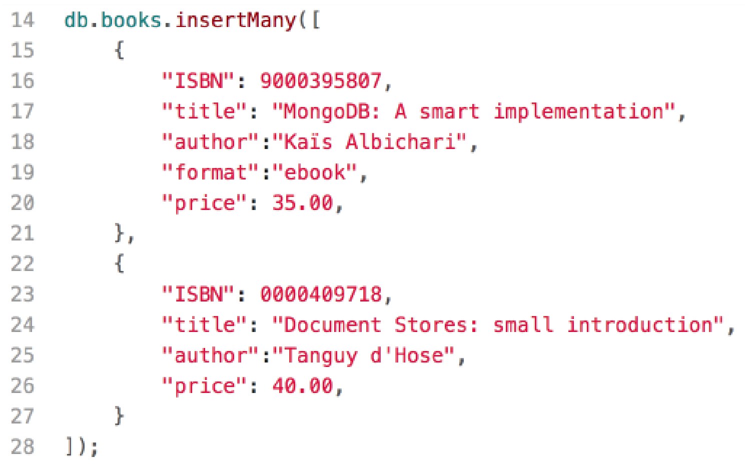
If a new book was to be added to the books collection, the following MongoDB operation would handle it:



```
db.books.inserOne(
    {
        "ISBN": 9000395807,
        "title": "MongoDB: A smart implementation",
        "author":"Kaïs Albichari",
        "format":"ebook",
        "price": 35.00,
    }
);
```

Figure 8: MongoDB’s insertion of a record

If several new books were to be added, the *insertMany* function would be the one to handle the operation:



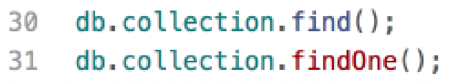
```
db.books.insertMany([
    {
        "ISBN": 9000395807,
        "title": "MongoDB: A smart implementation",
        "author":"Kaïs Albichari",
        "format":"ebook",
        "price": 35.00,
    },
    {
        "ISBN": 0000409718,
        "title": "Document Stores: small introduction",
        "author":"Tanguy d'Hose",
        "price": 40.00,
    }
]);
```

Figure 9: MongoDB’s insertion of a record

All writes are atomic at the level of a single document. Figure 9 is also a perfect example of loosened restrictions as the second inserted book does not specify the format.

4.4.2. Read Operation

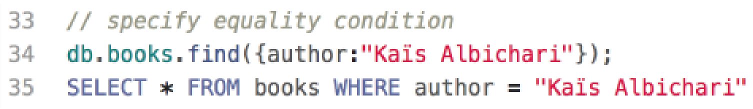
The read operation will retrieve a document from a collection. In other words: it queries a collection for a specific document. MongoDB implemented the following methods to do so:



```
db.collection.find();
db.collection.findOne();
```

Figure 10: MongoDB’s read commands

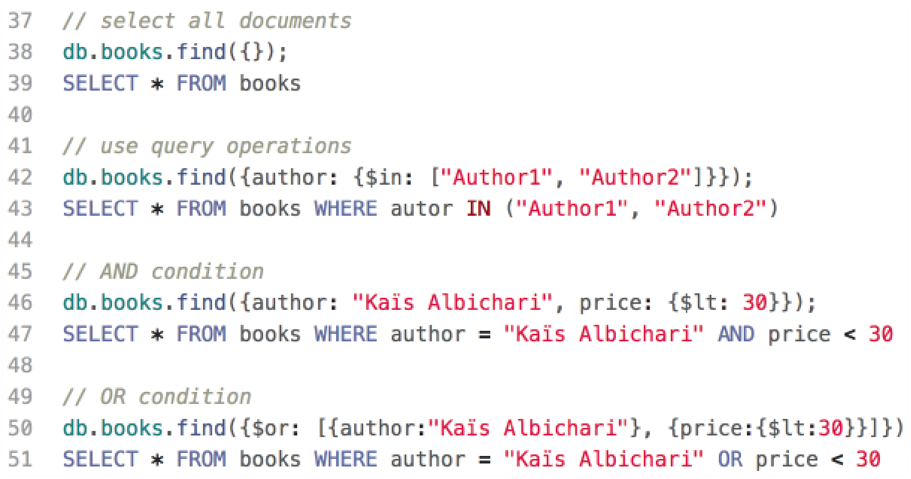
As expected, criteria or query filters can be specified in order to identify the documents that should be returned. Thus, querying for a book written by Kaïs Albichari can be done by executing the following operation:



```
// specify equality condition
db.books.find({author:"Kaïs Albichari"});
SELECT * FROM books WHERE author = "Kaïs Albichari"
```

Figure 11: MongoDB’s specific read commands

If only one result was required, the use of the findOne method is more appropriate. MongoDB has lots of possibilities when it comes to specifying criteria:



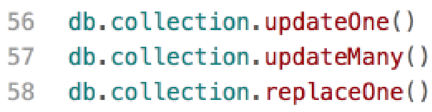
```
// select all documents
db.books.find({});
SELECT * FROM books

// use query operations
db.books.find({author: {$in: ["Author1", "Author2"]}});
SELECT * FROM books WHERE autor IN ("Author1", "Author2")

// AND condition
db.books.find({author: "Kaïs Albichari", price: {$lt: 30}});
SELECT * FROM books WHERE author = "Kaïs Albichari" AND price < 30

// OR condition
db.books.find({$or: [{author:"Kaïs Albichari"}, {price:{$lt:30}}]})
SELECT * FROM books WHERE author = "Kaïs Albichari" OR price < 30
```

Figure 12: MongoDB’s specific read commands

The examples above show that MongoDB can execute the same operations as the classical SQL language. As there are a lot of possibilities for different query types and operations, this document will not go further into the details. Nonetheless, all possibilities are listed in MongoDB’s online documentation.

4.4.3. Update Operation

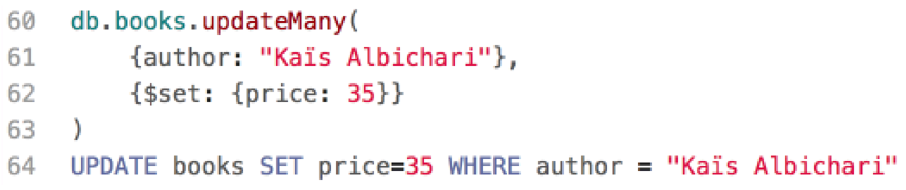
The update operation aims to modify the information of one or multiple records of the document store. MongoDB implements this operation trough three methods:



```
db.collection.updateOne()
db.collection.updateMany()
db.collection.replaceOne()
```

Figure 13: MongoDB’s update commands

The first two methods represent classical updates similar to the SQL INSERT command. The last method can be considered as a shortcut for deleting a record and replacing it by a new record. The syntax of these operations follows the conventions established for the previous operations:



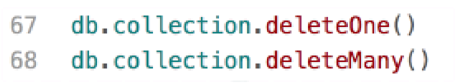
```
db.books.updateMany(
    {author: "Kaïs Albichari"},
    {$set: {price: 35}}
)
UPDATE books SET price=35 WHERE author = "Kaïs Albichari"
```

Figure 14: MongoDB’s specific update commands

Figure 14 shows how to update all books written by the same author. Interestingly, MongoDB’s methods are very readable and easy to use.

4.4.4. Delete Operation

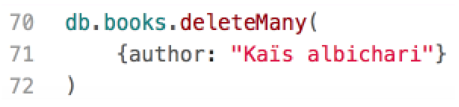
As usual, MongoDB implements multiple methods for the deletion of records. The two methods in charge of deleting records in MongoDB are:



```
db.collection.deleteOne()
db.collection.deleteMany()
```

Figure 15: MongoDB's delete commands

When deleting all records concerning a book of a same author, the command will be as follows:



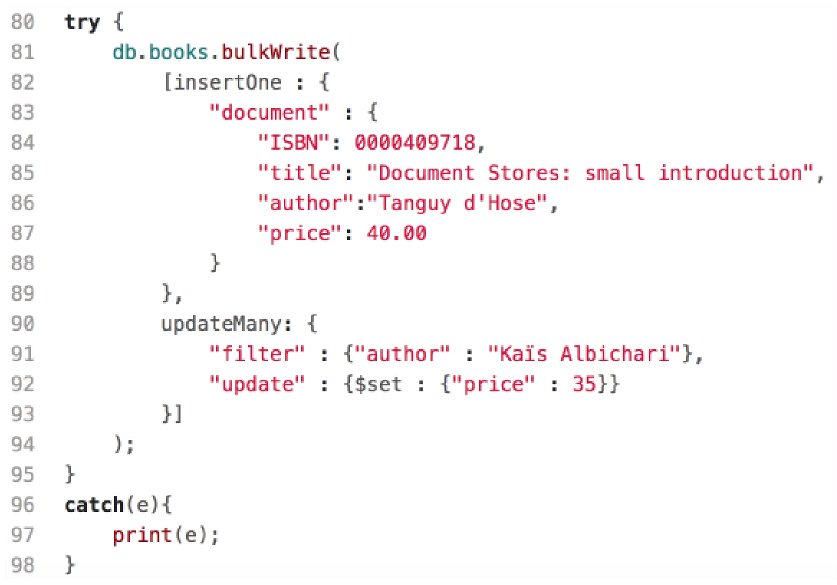
```
db.books.deleteMany(
    {author: "Kaïs albichari"}
)
```

Figure 16: MongoDB's specific delete command

4.5. Particularities

A particularity of MongoDB is its capacity to execute bulk actions. Some simple examples of these kinds of actions are the *insertMany*, *updateMany* and *deleteMany* methods. Of course, SQL also allows multiple updates and deletions using one operation. MongoDB surpasses these standard operations by implementing the *bulkWrite* method.

This method enables the user to specify multiple types of operations, execute them simultaneously and get a feedback.



```
try {
    db.books.bulkWrite(
        [insertOne : {
            "document" : {
                "ISBN": 0000409718,
                "title": "Document Stores: small introduction",
                "author":"Tanguy d'Hose",
                "price": 40.00
            }
        },
        updateMany: {
            "filter" : {"author" : "Kaïs Albichari"},
            "update" : {$set : {"price" : 35}}
        }]
    );
}
catch(e){
    print(e);
}
```

Figure 17: MongoDB's bulkwrite command

This method generates a feedback with all information regarding the number of updates, deletions, insertions, etc.

5. Scenario using MongoDB

5.1. Scenario in action

5.1.1. Problem’s exposure

As mentioned previously, the startup named *Nahmazon* is using a classic relational database. However, the possible expansion of this startup, whether in international presence or inventory size, is a potential problem for the current SQL-based relational database management system.

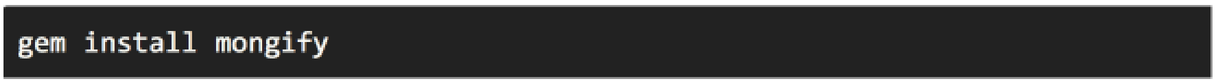
Entities could potentially change multiple times or be declined in multiple ways, and product-specific social features could be implemented. It becomes obvious that Nahmazon will be confronted to unstructured data in large quantities without necessarily requiring relationships between entities.

For that reason, we believe a non-relational model would fit better. A part from the strictly conceptual point of view, the expansion of Nahmazon will require server-side investments. As relational models do not scale horizontally easily, the company will have to invest in expensive servers. If the company opts for a NoSQL approach, horizontal scaling becomes the best possibility. Popular DBMSs such as MongoDB allow the server to be divided in pools where more affordable machines could manage only some of the data. Consequently, MongoDB would be a perfect fit for the company.

5.1.2. Migration from SQL data to MongoDB

It is crucial the problem is handled before the startup really grows. Therefore, we suggest a migration from the SQL infrastructure to an equivalent MongoDB setup. For this purpose, we use a software called *Mongify*. This software makes migrations from SQL to Mongo something trivial and well executed.

To install the software, simply open the terminal and run following command:



```
gem install mongify
```

Figure 18: Mongify install command

Once the installation is completed, open the database.config file to create the connection for both the relational database and MongoDB. The command “mongify check -c database.config” checks the connection of the SQL and the NoSQL databases.

The most important part of the process is the command “mongify translation -c database.config”. It will obviously translate the structure of the SQL database into a MongoDB structure.

It is interesting to pipe the translation into a file with the command “mongify translation -c database.config > ‘translation_file’”. Having this file allows the user to specify some behavior he’d like to have in the MongoDB database, such as an equivalent of foreign keys.

As previously mentioned, the NoSQL model does not work with keys. However, the startup’s current database is managed using SQL. In other words, the current infrastructure relies on keys. For that reason, during the translation, the user can specify in which place some value has to be the same as another.

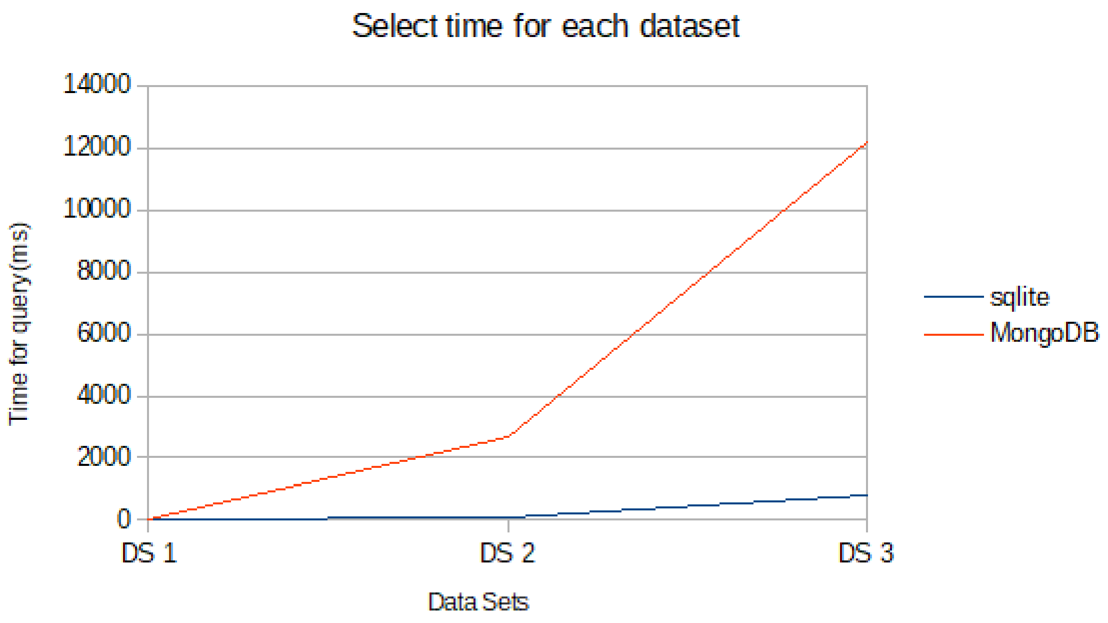
The final command is “mongify process ‘translation_file’ -c database config”. After these 3 commands, and possibly some modifications in the translation file, the migration is completed. From now on, *Nahmazon* can start working with this brand-new database.

5.1.3. Comparison SQL – MongoDB

Some runs have been set to check the viability of the proposed solution. At this end, let’s consider 3 data sets DS1, DS2, DS3 of 500, 5000 and 50 000 records respectively.

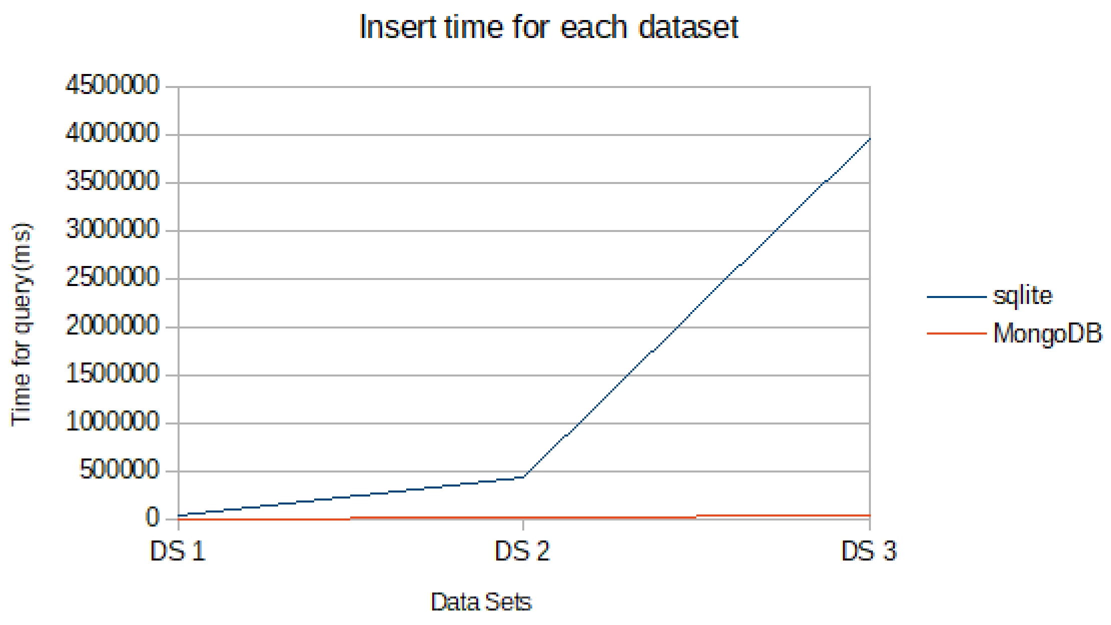
5.1.3.1. Select operation

Data Sets	sqlite	MongoDB
DS 1	12 ms	15 ms
DS 2	66 ms	2681 ms
DS 3	798 ms	12173 ms



5.1.3.2. Insert operation

Data Sets	Sqlite	MongoDB
DS 1	34609	538
DS 2	422731	4122
DS 3	3947183	42696



5.1.3.3. Deductions

The graphs show that these are two inverse behavior, where sqlite is faster for a great amount of selections, while MongoDB is faster with insertions.
However, for DS2 and DS3, the larger sets, the execution time for a selection with MongoDB is better than the execution time for an insertion with sqlite.
As the startup wishes to expand his activity, a great amount of incoming data is predictable.
It goes without saying that the insertions of these data won't be a problem with MongoDB. With the startup expanding, there will be a huge number of insertions, which would take too much time with sqlite.

5.1.4. Current scenario : SQL vs MongoDB

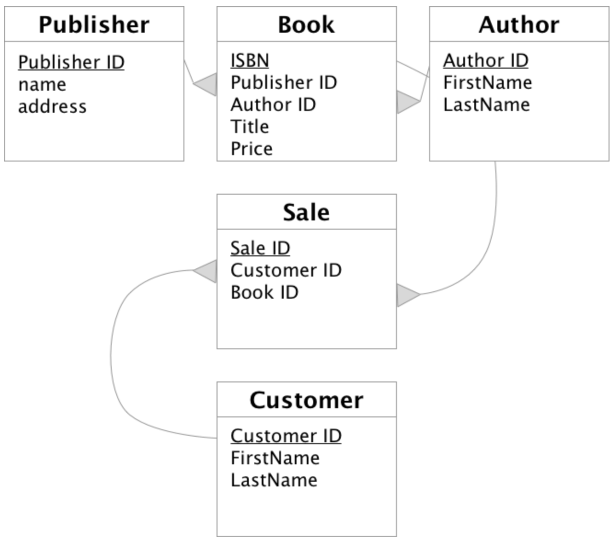


Figure 19: current infrastructure in SQL

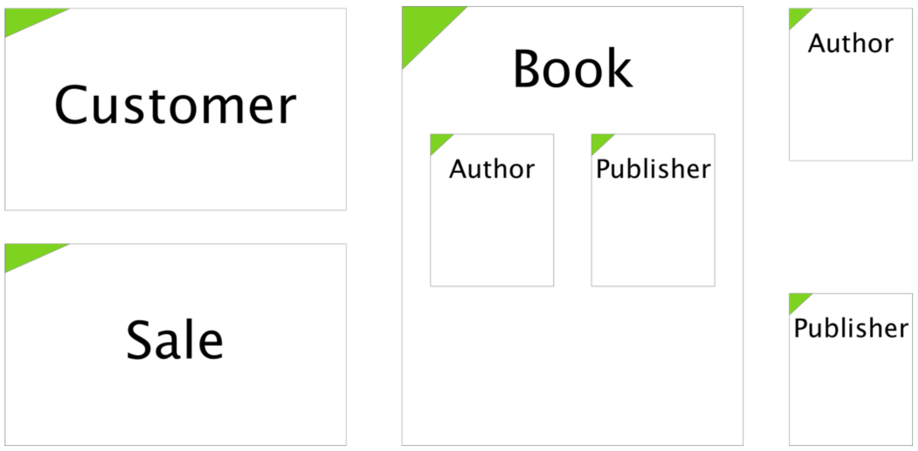


Figure 20: current infrastructure in MongoDB

Though differences may seem small between the two infrastructures, the real structural need for MongoDB is still unclear as the differences are quite small. Fortunately, the next section clearly depicts why MongoDB is the best choice for Nahmazon.

5.1.5. Future scenario: SQL vs MongoDB

As mentioned previously, the startup's ambition to develop and expand their current inventory is one of the elements that has led to the migration to MongoDB. For instance, the Sale table in the SQL structure is using a foreign key pointing to a book record. This means that there will always be a field pointing to the Book table. Unless all the future products the startup will sell in the future are already present in the database, the Sale table will have to be modified to include each new product. As a result, the sale table will be enormous and full of None values.

When using MongoDB, the Sale collection hasn't any constraint concerning its fields. The variety of recorded products can grow without the needing to change the collection's structure: the field currently used for the Books, could easily be used for any other product.

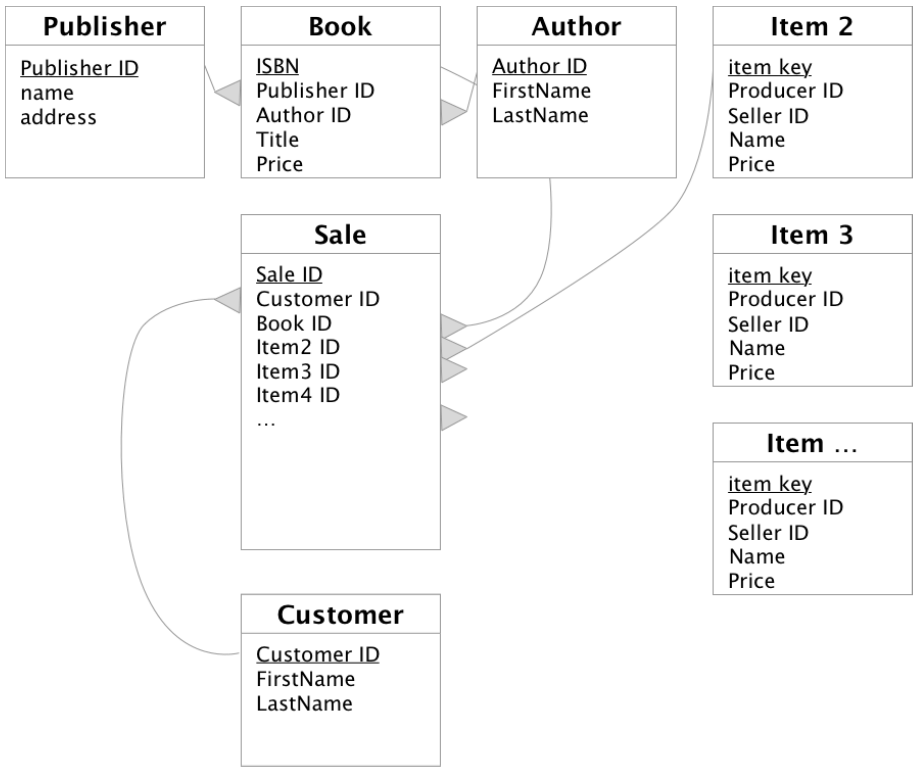


Figure 21: partial future infrastructure in SQL

When designing a future infrastructure in SQL, different entities make everything more complicated: all entities have their own structure and keys. As Nahmazon clearly stated it would like to increase ease of management, such a infrastructure wouldn't be compatible with their possible expansion.

When replicating a similar infrastructure using MongoDB, we quickly notice that the fact that it accepts unstructured data simplifies the architecture.

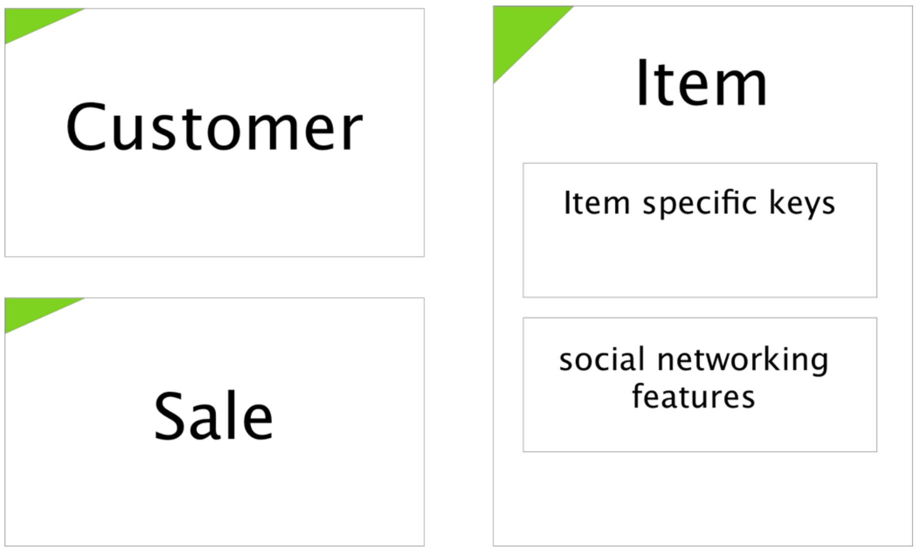


Figure 22: partial future infrastructure in MongoDB

When using MongoDB, all items of the inventory can be generalized to an Item without having to consider each item's particularities. As data is unstructured, all item specific keys and additional features can also be customized case by case. All in all, apart from scaling, security trough *sharding* and ease of management, MongoDB also offers ease of structural design.

5.2. MongoDB advantages and disadvantages

5.2.1. Pros

MongoDB offers several advantages. First of all, the fact that MongoDB is free and open source enables any startup to use it and run it on a Linux server. The budget that would otherwise have gone into DBMS-costs can now be used for other needs of the company.

Next, the schema-less aspect of MongoDB is perfectly adapted for an eternally changing platform. This enables the company reconsider its infrastructure at any time and always have a database structure that corresponds to their needs.

The horizontal scalability perspectives offered by MongoDB is hugely interesting for a company such as Nahmazon. Trough sharding, MongoDB makes it possible to go beyond hardware limitations and distributes the data across multiple physical partitions. By adding more machines, the working set will beneficiate from more RAM.

Finally, MongoDB supports replica sets. This means that it is capable to handle the failover mechanism. If the primary server goes down for any reason, the secondary server takes the upper hand and becomes the primary server. As MongoDB automatically handles this, no human intervention is required.

5.2.2. Cons

First of all, due to denormalization and the presence of field names in each document, the size of the database will be larger than when using a relational model.

Next, the absence of JOIN operator makes queries less flexible. Even though denormalization is able to handle JOIN-like requests, update queries will be slower.

Finally, the available RAM is what determines to limitations of the database. If the database grows in a way that wasn't predictable, the lack of RAM could lead to failed insertions, without any warning.

6. Conclusion

Concerning the objective, it is obvious Nahmazon needs to change its infrastructure and move its data to MongoDB. The features offered by MongoDB fulfil all their requirements, making it the best option for comfortable data management and company expansion.

Concerning the project, this project has been really useful to discover the wide variety of database paradigms and DBMSs. The research of NoSQL, document stores and MongoDB has clearly shown us that there is much more than only relational databases and SQL.

Aside from opening our perspectives on the world of databases, this project has shown us that research is important when building software: it is important to know what is to be achieved and expected, but also know about all the constraints and ultimately find the best suiting infrastructure

7. Bibliography

Amazon Web Services, Inc. (2017). *Présentation de NoSQL – Amazon Web Services (AWS)*. [online] Available at: https://aws.amazon.com/fr/nosql/?nc1=h_ls [Accessed 1 Dec. 2017].

Basho. (2017). *Document Databases*. [online] Available at: http://basho.com/resources/document-databases/ [Accessed 1 Dec. 2017].

Basho. (2017). *NoSQL Databases*. [online] Available at: http://basho.com/resources/nosql-databases/ [Accessed 1 Dec. 2017].

Database.guide. (2017). *What is a Document Store Database? | Database.Guide*. [online] Available at: http://database.guide/what-is-a-document-store-database/ [Accessed 1 Dec. 2017].

Mongify.com. (2017). *Mongify - Move data from SQL to MongoDB with ease - Transforming your data from sql to MongoDB in a few simple steps - Mongify.com*. [online] Available at: http://mongify.com [Accessed 18 Dec. 2017].

MongoDB. (2017). *Document Databases*. [online] Available at: https://www.mongodb.com/document-databases [Accessed 1 Dec. 2017].

MongoDB. (2017). *NoSQL Databases Explained*. [online] Available at: https://www.mongodb.com/nosql-explained [Accessed 1 Dec. 2017].

MongoDB. (2017). *What Is MongoDB?*. [online] Available at: https://www.mongodb.com/what-is-mongodb [Accessed 1 Dec. 2017].

Msdn.microsoft.com. (2017). *Data Points - What the Heck Are Document Databases?*. [online] Available at: https://msdn.microsoft.com/en-us/magazine/hh547103.aspx [Accessed 1 Dec. 2017].

SearchDataManagement. (2017). *How to determine which NoSQL DBMS best fits your needs*. [online] Available at: http://searchdatamanagement.techtarget.com/feature/How-to-determine-which-NoSQL-DBMS-best-fits-your-needs [Accessed 1 Dec. 2017].

Sisense. (2017). *Postgresql vs. MongoDB for Storing JSON Database* | *Sisense*. [online] Available at: https://www.sisense.com/blog/postgres-vs-mongodb-for-storing-json-data/ [Accessed 1 Dec. 2017].