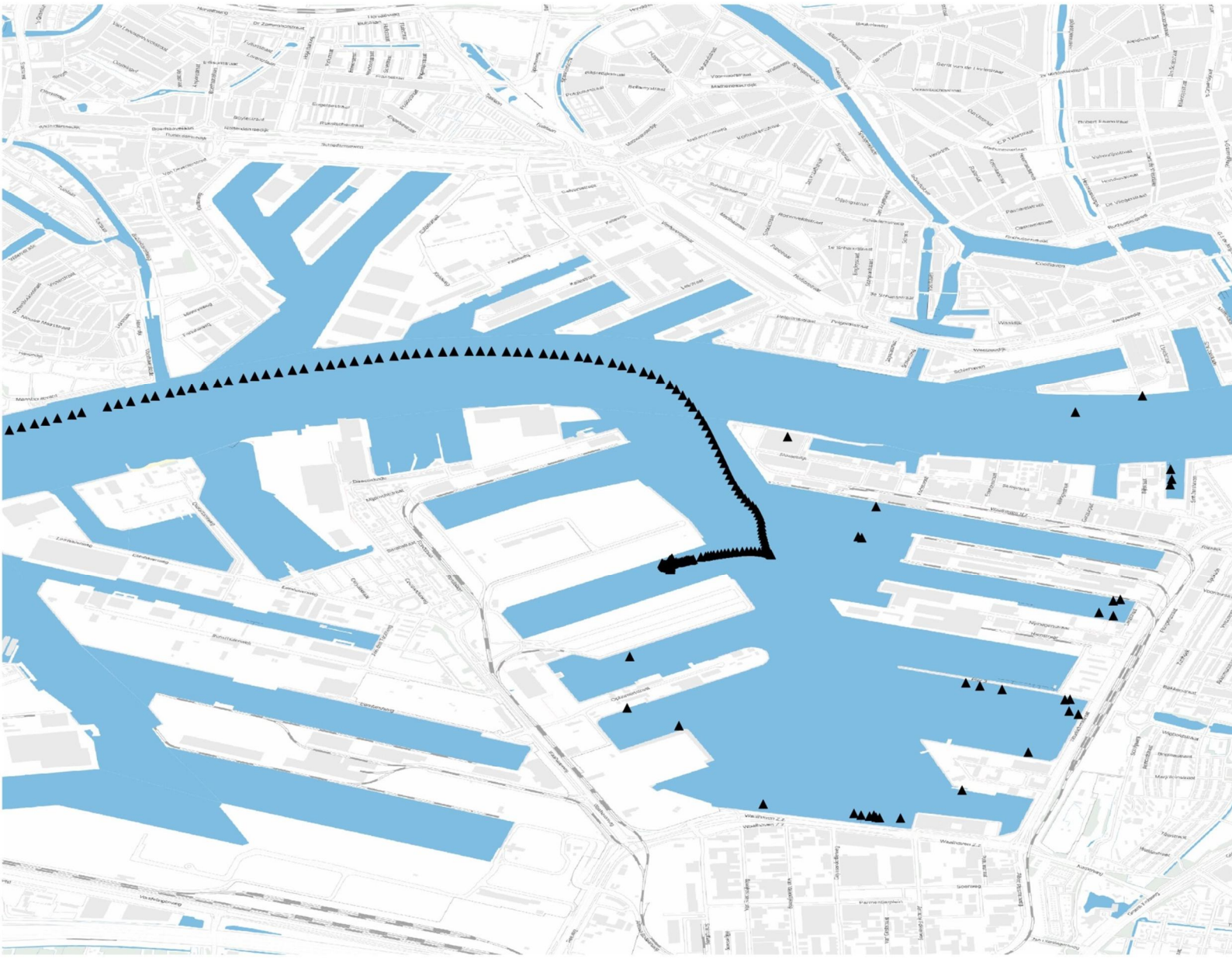

Master of Science Geomatics

Managing Historic Automatic Identification System data by using a proper Database Management System structure

Irene de Vreede

November 2016

TU Delft

Rijkswaterstaat
Ministerie van Infrastructuur en Milieu

MANAGING HISTORIC AUTOMATIC IDENTIFICATION SYSTEM DATA BY USING A PROPER DATABASE MANAGEMENT SYSTEM STRUCTURE

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of

Master of Science in Geomatics

by

Irene de Vreede

November 2016

The work in this thesis was made in:

Delft University of Technology
Department of OTB - Research for the Built Environment
Faculty of Architecture and the Built Environment

Rijkswaterstaat

Supervisors:	C.W. Quak
	Prof. dr. ir. P.J. M. van Oosterom
Co-reader:	J.W. Konings

ACKNOWLEDGEMENTS

This MSc thesis is the result of several months of hard work in association with the support of many people.
First and foremost i would like to thank my supervisors Wilko Quak and Peter van Oosterom for their guidance and inspiring ideas throughout the whole research. Special thanks goes out to Wiebrand Bouwkamp and Brian Langenberg from Rijkswaterstaat, for introducing me to the AIS data and the daily processes within Rijkswaterstaat. Thanks to Martijn Meijers for his help with the decoding of the AIS data which made the data usable and this research possible. My gratitude goes also to my co-reader Rob Konings and delegate of the Board of Examiners Marietta Haffner for their useful questions and feedback.
Then i would like to thank my fellow students Adrie and Pieter for the inspiring and stimulating online conversations, discussions and encouragements.
And finally i would like to thank my parents, sisters, friends, the girls from my Waterpolo team and my boyfriend Bruno for supporting me and believing in me.

ABSTRACT

This MSc thesis aims to develop a management solution for historical Automatic Identification Structure (AIS) data by finding a suitable database and data(base) structure for an effective extraction of historical AIS data to support Spatial-Temporal analyses. The work-flow the thesis proposes focusses on the selection of a database, the organization of the data inside this database and the effectiveness of the extraction of information that can be used in Spatial-Temporal analyses.

AIS data, vessel movement data, has a complex structure consisting of 27 different kind of encoded messages. These messages hold their own unique and similar data and are related to each other by their vessel-ID number (MMSI). The update rate of the messages, the data, is dynamic, different for each kind of message. Storing and structuring historic AIS data should be able to handle the data features.

Based on the comparison between the AIS data features, the requirements that the database has to meet which are set by Rijkswaterstaat and related literature and documentation is MongoDB a suitable database to manage AIS data. The requirements that are set by Rijkswaterstaat are based on three use cases; location, trajectory and bounding box. Extraction of the information, that can answer the use cases, from the historic AIS data that is stored within the database, should be effective.

To ensure this, two data(base) organizations are developed, one storing AIS data in a way that it will support the Spatial-Temporal analyses of the specified use cases and one storing AIS data in a way that it will support all possible spatial-temporal use cases. The first will store only the four decoded attributes that are necessary to answer the use cases together with the original encoded AIS message. The second will store all decoded attributes an AIS message holds.
To enhance the performance of MongoDB holding AIS data in one of the two data(base) organization, four indexes on individual attributes and one 4D Morton code index are developed. A 4D Morton code index based on latitude, longitude, MMSI and date-time is developed to enhance the effectiveness of the extraction of the information from the database that will answer the three use cases.

A comparison between the two data(base) organizations and the five indexes by measuring their effectiveness while executing three, on the by Rijkswaterstaat specified use cases based, queries, concludes on a suitable management solution for historical AIS data which will support Spatial-Temporal analyses.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AIS Automatic Identification System xi
API Application Programming Interface 43
BSON Binary JSON 24
CIV Centrale Informatie Voorziening 16
CPU Central Processing Unit 17
CSTDMA Carrier Sense Time Division Multiple Access 4
GPS Global Positioning System 3
K-NN K-Nearest Neighbours 4
MMSI Maritime Mobile Service Identity-number 23
MOD Moving Objects Databases 9
MOST Moving Objects Spatial-Temporal 7
RAM Random Access Memory 51
SFC Space Filling Curve(s) xi
SOTDMA Self-Organized Time Division Multiple Access 4
SQL Structured Query Language 22
TDMA Time Division Multiple Access 4
UML Unified Modelling Language 17
VTS Vessel Traffic Service 3
YCSB Yahoo! Cloud Serving Benchmarking 10

1 INTRODUCTION

'We are drowning in information, but starving for knowledge'

The claim made by Naisbitt and Cracknell [1982] is nowadays relevant to how is being dealt with Automatic Identification System (AIS) data.

1.1 BACKGROUND

AIS enables the identification, tracking and monitoring of vessels. It is the most frequently used marine system around the world. This system, AIS, provides rich real-time vessel movement data and information to identify other vessels, and provides extra information including vessel name and number, the speed of the vessel, the destination etc. AIS is widely used around the world to avoid vessel collisions, for security, for efficiency of navigation, for communication between vessels and between vessel and shore, for information and for safety. AIS is mainly used by Vessel Traffic Service (VTS) due to the provision of real-time positions and other interesting information of surrounding vessels.

AIS data consist of digital messages which vessels and base stations sent to each other. Vessels are able to sent 27 different kind of AIS messages all **NMEA-0183 encrypted** [1] and containing unique and similar data. Where as certain messages contain the vessel position (dynamic information), others contain the vessels static and voyage related information such as the name and destination. There is not one message that contains all available information of a vessel.
The data these AIS messages contain are generated by sensors, Global Positioning System (GPS) or are manually provided by the vessels owner. All sea-going vessels larger than 300 gross tons, all commercial and recreational vessels larger than 20 meters, and **CEMT** [2] class 1 or higher are according to the International Maritime Organization and Dutch regulations obliged to have an AIS system on board which consist of sensors, transponders, receivers and soft- and hardware[AISM and IALA, 2005; Tetreault, 2005].

Vessels sent their messages according to an update rate, which differs per message type. The update rate of AIS messages containing the vessels position depends on the speed of the vessel. The highest update rate for these messages is once every two seconds. For the messages containing static or voyage related data as ship name and destination, this update rate is much

1 **NMEA-0183** is a standardized data specification for communication between different marine electronics. Encrypted; decoding is necessary to extract the information these NMEA 0183 encrypted messages contain

2 **CEMT**, Conference Euopeenne des Ministres de Transport define classes for vessels, Vessels are divided in classes based on the dimensions of the vessel

lower, once every 3 to 6 minutes depending on if the vessel is moving.
AIS data is sent from vessels, shore to vessels, and shore across two maritime mobile radio frequencies. Two channels (161.976MHz and 162.025MHz) are used to avoid interference problems. Each channel allows multiple vessels to sent their messages. Sharing a channel with multiple users is done with the use of a Time Division Multiple Access (TDMA) scheme. This TDMA scheme divides the signal into time slots. On each of the two channels there are 2250 time slots available per 60 seconds for each base station. Vessels sent their messages to one of these time slots. Each time slot can contain exactly one message. The way time slots are used depends on the class of AIS system on board of the vessel. Class A AIS systems reserve space in the slot map and messages are sent into these reserved time slots (Self-Organized Time Division Multiple Access (SOTDMA)). Class B AIS systems scan for available space in the slot map and their messages are sent towards the found free time slots (Carrier Sense Time Division Multiple Access (CSTDMA)).
It might be the case that more messages are sent to the slot map than available time slots. Then the closest vessels are more likely to be transmitted since messages from vessels that are further away are possibly sent to a timeslot of another base station as well.

1.2 PROBLEM STATEMENT

AIS is originally developed to support VTS, this is why AIS's main use is by VTS. For VTS, all different AIS messages are important, especially the ones containing vessel positions, safety related messages or messages containing information on for example dangerous cargo. Of course only the most recent messages, positions and information are used since what happens right now on the waterways is most important. This means that 'old' messages, positions and information is no longer relevant for VTS and is often 'deleted' from memory. This deletion of data is a great loss since the 'old' positions, the historic AIS data, is very useful for many different applications.
Ristic et al. [2008]; Wijaya and Nakamura [2013] for example examine the possibility to use AIS data for ship movement behaviour prediction. Ristic et al. [2008] uses statistical analysis of AIS data for the detection of possible anomalies in vessels motions. When the normal behaviour of a vessel is assumed, a predication of future vessel motions shall be made. Wijaya and Nakamura [2013] predict the future position of a vessel by finding the K-Nearest Neighbours (K-NN) from historical positions of vessels which are of similar type, have a similar navigational status and draught that have already visited the current position of the vessel of interest. The research of Wang et al. [2008] focusses on anomaly detection within vessels movement.

The above mentioned studies have especially focused on the use of AIS data, they neglect the storage of the data prior to the ability to perform statistical or other kinds of analyses. Related studies focussing on **moving objects**[3], do cover this part. Hecht and Jablonski [2011]; Hussien [2012]; Indrawan-Santiago [2012]; Moniruzzaman and Hossain [2013]; Wolfson et al. [1998] examine the subject of using databases for the storage of moving objects data and the existence of multiple different kinds and types of databases which have features that others do or might not have. Chen et al. [2008a];

3 **Moving objects** can be defined as people, animals, cars, plains, vessels and other "objects" that change their position in relation to time

Meng and Chen [2011]; Mokbel et al. [2003b]; Nguyen-Dinh et al. [2010]; Park et al. [2013] elaborate on the use of databases for the storage of AIS data by their researches on storage performance optimization methods for efficient management of moving objects.
The moving objects data that these researches apply, often consist of or focus on just the latitude, longitude and time. AIS data contains much more interesting and important data than these four attributes. Therefore the question arises; Will the methods and techniques provided by research on the management of moving objects be applicable and evenly sufficient for historical AIS data. This MSc thesis therefore will present a research towards a management solution for historic AIS data intended to encourage further spatial-temporal analysis. The method provided searches for a suitable database and develops a data(base) organization. The research questions this research will answer will be presented in section 1.3.

To come back at the statement of Naisbitt and Cracknell [1982] at the beginning of this introduction; the large amount of vessels equipped with an AIS will, when storing this data, let us eventually drown in information due to the large amount of vessels (in Europe there are already around 21.000 vessels) and the high update rate of the data. Knowledge on how to manage this historic AIS data, though, remains quite unknown.

1.3 RESEARCH QUESTIONS

This MSc thesis aims to provide a management, storage, and structuring solution for historic AIS data to support spatial-temporal analysis. The following question will be answered:

How can historic AIS data be managed, stored, and structured to support spatial-temporal data analyses?

To answer the main question five sub questions have been specified:

- What is AIS data, what are its features?
- What kind of spatial-temporal analyses with historic AIS data is interesting (for Rijkswaterstaat)?
- What database should store historic AIS data?
- How should this database store the historic AIS data?
- Which indexing technique is suitable to provide efficient historic spatial-temporal data requests?

1.4 MOTIVATION

There is great demand for AIS data available by government, companies and researchers. Providing a solution for the management of historic AIS data will provide an answer to this demand, the possibility to use the data for multiple different applications and spatial-temporal analysis.
The development of applications that would benefit from the storage of moving objects make this topic widely researched. The fact that this data can be

used for many different purposes is becoming more well-known, thence the rising interest in the topic of moving objects data management. The focus, though, seems to be on real-time applications and data management where seconds 'old' data is not relevant. The management of historical moving objects does however deserves attention. This research on the management of historical AIS data aims at giving this topic the attention. Storing moving objects is indicated as difficult due to the fact that the data needs to be kept up-to-date. The management of AIS data is indicated as not yet widely researched due to researches focussed on the use of AIS data and therefore are negligent on how to store it. This MSc research aims at bridging these gaps of the management of historical moving objects data and AIS data.

1.5 SCOPE

This research is, as the main research question already implies, focussed on the management of historic AIS data. The real-time update of AIS data will be taken into account when developing a suitable cluster or index technique but it will not be tested in this research.
The kind of spatial-temporal data analysis for which an AIS management structure will be defined are described by different people of the CIV department of Rijkswaterstaat.
The used AIS data within this research is delivered by Rijkswaterstaat and consists of AIS data from the Dutch waterways only.
Privacy is the main reason why AIS data is not efficiently stored. The privacy issues concerning the storage of AIS data are not taken into account in this research, though this research will properly deal with this privacy sensitive data and will therefore not publish information that can be related to pivate matters.

1.6 THESIS OUTLINE

Chapter 2 presents the related work. Available research related to the management, storing and structuring of moving objects data is discussed in this chapter. Chapter 3 describes the aspects of the methodology that is used to create a suitable solution for the management; storing and structuring of AIS data. The chapters 4, 5 and 6 then will explain the implementation of the followed methodology. Where chapter 7 presents the results and analysis of the executed tests. Chapter 8 then concludes on the AIS data management solution which will support spatial-temporal analysis. This chapter will also give directions for future work.

2 RELATED WORK

Several areas of research are relevant for this thesis. The following chapter provides an overview of the related works. The first part, section 2.1 will address the researches related to the management; storing and structuring of moving objects. A clear distinction existing in this researches is indicated and the researches on data management of historical positions of moving objects will be highlighted. The second part, section 2.2 is focussed on the accessible tools for the management and section 2.3 will explain the different possibilities for an efficient storage and structuring of a database for the management of historical positions of moving objects, namely spatial-temporal access methods and Section 2.4 will discuss comparing different indexing, spatial access methods. The chapter will end with queries that are most likely to be asked to databases storing moving objects (section 6.2.2)

2.1 MOVING OBJECTS DATA MANAGEMENT

Moving objects data management is in this research defined as the storage and structuring of data from moving objects (vessels) within a database to support spatial-temporal data analyses. Moving objects are defined as people, animals, cars, plains, vessels and other "objects" that change their position in relation to time. They can be classified into moving points and moving regions. Moving points are objects that change location over time, moving regions are objects that change shape/extent over time [Guting and Schneider, 2005]. The data that can be retrieved from moving objects is at least a 2D position in combination with a time stamp. Moving objects, points or regions, therefore can be referred to as multi-dimensional spatial-temporal data.
Various studies in recent years their focus on moving objects and related models and algorithms. In general, these studies are interpreted from two perspectives [Guting and Schneider, 2005; Meng and Chen, 2011]. The studies on this subject are divided in the two. The first perspective is focussed on the representation of the current real-time changing positions of moving objects and their possible future positions. The second perspective is focussed on the representation of the historic positions of moving objects [Pfoser et al., 2000; Reiss et al., 2007; Diallo et al., 2015; Frentzos, 2008].

With the focus on current and future positions several algorithms and models are proposed by various researches. Sistla et al. [1997] proposes the Moving Objects Spatial-Temporal (MOST) data model. In this model the position of a moving object is represented as a function of time. Moving objects are referred to as dynamic attributes which change over time even if its position is not updated. This means that an assumption is made to the future position of moving objects. This makes it possible to ask queries that refer to the future values of these objects.

Another model is proposed by Chon et al. [2001], the space-time grid model [Meng and Chen, 2011]. The space-time grid model of states that moving objects are existing in two-dimensional space on a one-dimensional line (linear network). This line for example can be defined as a highway with a specified amount of lanes and a specified amount of cars it can accommodate before traffic jams occur. These constraints are developed for an efficient traffic management. Management of the moving objects in this model is performed by storing the latest arrived positions of moving objects that fit within a predefined time domain, and meet the constraints, in primary storage. When data is received that has a time stamp beyond the predefined time domain, the time domain will shift forward and the positions that do not fit into the time domain any more will be stored into secondary storage or not even stored at all.
Meng and Chen [2011] proposes two more models for the management of current and future positions of moving objects, the abstract data types model Lema et al. [2003] and the linear constraint data model.

2.1.1 The management of historic positions

The management of historic moving objects data for the purpose of spatial-temporal analysis is often mentioned in research to indicate the existing division in researches. Often though these researches continue with the investigation towards management solutions for current and future positions [Brakatsoulas et al., 2004; Chon et al., 2001; Guting and Schneider, 2005; Lema et al., 2003; Meng and Chen, 2011]. The management and use of real-time position data imply above all, that (seconds) 'old' position data is not necessary and can be deleted or don't have to be stored at all.
Performing spatial-temporal analysis on moving objects data though pleads for the storage of this 'old' data.

Reiss et al. [2007] and Nittel [2015] imply the management of historic positions to be difficult due to the fact that the historical data needs to be up-to-date. Holding historical data up-to-date means updates of new data every other second depending on the update rate. Management of historical data therefore is associated with the management of real-time data by the (real-time) update rate.
Reiss et al. [2007] approaches the management of historical data from a data-stream point of view and implies on a solution to manage and query current and historical data at the same time by using bitmap indices. Bitmap indices is a kind of index that compactly stores the data in **bits**[1]. Reiss et al. [2007] concludes that historical positions of moving objects will not be modified, only new positions are appended to the existing positions. Therefore bitmap indices do not have to be sorted, and new records can be easily added.

2.2 DATABASES AS THE PROPER MANAGEMENT TOOL

Databases are tools mainly used to efficiently store, structure and retrieve again all kinds of data. Data that is most commonly structured. **Tradi-**

1 **Bit** or binary digit is a basic unit of information within the computing world. They consist of a 0 or a 1

tional Relational Databases [2] have been leading within the data management, these systems were supposed to be the solution for almost all data management problems.
The growing amount of data that is acquired these days is becoming more complex. Storing and analysing this large volumes of complex data while using a traditional relational database will lead to problems with the data modelling and **horizontal scaling** [3] [Moniruzzaman and Hossain, 2013]. The growing volume and complexity of data have pushed the traditional relational databases to their limits. This has had an effect on the amount of organizations building their own databases suitable for their needs. These databases were later defined as **NoSQL databases** [4] [Lourencco et al., 2015; Moniruzzaman and Hossain, 2013].
The storage of the positions and other data of moving objects proposes other challenges (besides the growing volume and complexity of data) to the existing traditional storage technology. Traditional relational databases are not well suited for the storage of moving objects due to the fact that these databases originally are designed for the storage of constant datasets or datasets with a low update rate since data always needs updates. They assume that the data is constant until it is explicitly modified, updated. The spatial-temporal nature of moving object data implies a dynamically fast growing infinite dataset of the objects positions combined with time [Rodríguez-Tastets, 2005]. Storing such data in a traditional relational database will impose a serious performance overhead since the location of the moving object is updated frequently [Wolfson et al., 1998].
According Hussien [2012] there are several capabilities that traditional relational databases do not have but are necessary for the management of moving objects. One of those capabilities is the dealing with the frequent updates of moving objects data as is explained earlier. Another is the combination of spatial and temporal entities which with retrieval of data from the database will be asked together. This has led to the development of Moving Objects Databases (MOD) or spatial-temporal databases. These kind of databases often reside to spatial databases which include a temporal element or temporal databases that try to include a spatial component [Guting and Schneider, 2005]. The scalability problem traditional relational databases face have not yet been solved with the development of spatial-temporal databases of which the base relies on the relational databases.
NoSQL databases were thought off to be the solution to the limitations the traditional relational database systems encountered. NoSQL databases though are often specialized for specific needs, the specific needs of a company. This makes the selection of one most suited for a use case difficult [Abramova et al., 2014; Hecht and Jablonski, 2011].

2.2.1 Database benchmarking

The rising interest in NoSQL databases does not mean that such a database is the best suited for all use cases. Each database offers different solutions for specific cases. For example, just a few databases could be suitable for the storage of spatial-temporal data since they have a geospatial extension [Baas, 2012]. Depending on the use case there should always be one database

2 **Traditional Relational Databases** are databases which hold a homogenous data set with existing relationships among the data, they represent the data as a collection of relations
3 **Horizontal scaling** means scaling by adding more machines to the resources
4 **NoSQL** databases are containing different features than traditional relational databases have

that outperforms the others, therefore there does not exist one database that fits all cases [Lourencco et al., 2015]. The presence of many different database benchmarking, database comparison researches as Kashyap et al. [2013]; Hecht and Jablonski [2011]; Indrawan-Santiago [2012], confirms this non-existence of one database that fits all use cases. These database benchmarking researches can divided into three groups, researches that compare only NoSQL databases Kashyap et al. [2013]; Abramova et al. [2014]; Hecht and Jablonski [2011], researches that compare NoSQL databases with traditional relational databases Parker et al. [2013]; Li and Manoharan [2013] and researches that compare Traditional relational databases.

Several studies use while comparing different databases the Yahoo! Cloud Serving Benchmarking (YCSB) tool. This tool is set as a standard for comparing databases and randomly generates the data and the operations that has to be carried out [Kashyap et al., 2013]. Using this YCSB tool by comparing databases, the outcome should theoretically be applicable to all cases. The conclusions drawn from researches though often mention the necessity of database comparison based on certain use cases [Hecht and Jablonski, 2011; Klein et al., 2015]. Klein et al. [2015] itself proposes a framework for finding the best suitable database for a use case.

2.3 HISTORICAL POSITIONS DATA(BASE) ORGANIZATION OPTIMIZATION

The development of efficient index structures is an important issue. Efficient indexing and clustering techniques are effective optimization methods for these databases storing spatial-temporal or any other kind of data. They are important for efficient execution of **queries** [5] involving spatial-temporal constraints [Park et al., 2013].

2.3.1 Spatial-temporal indexing and cluster methods

There have been numerous researches in developing spatial-temporal indexing methods to support database independent spatial-temporal queries [Mokbel et al., 2003b; Nguyen-Dinh et al., 2010]. Mokbel et al. [2003b] describes different spatial-temporal indexing methods researched until 2003 and Nguyen-Dinh et al. [2010] continues the Mokbel et al. [2003b] research by covering spatial-temporal indexing methods published after the year 2003. It explains new methods and addresses the weaknesses of existing methods. Both Mokbel et al. [2003b]; Nguyen-Dinh et al. [2010] give methods for indexing the 'historic' position, methods for indexing the current positions and methods for indexing possible future positions.

Indexing the multi-dimensional past

Traditionally spatial-temporal data is indexed using an R-tree or a variant. The R-tree can be defined as a height balanced indexing structure for multi-dimensional data. It can organize any-dimensional data. Its main idea is to group objects that are close to each other and then represent this group

5 **Queries** are commands given to the database to perform certain actions, which returns potential data, a selection of data

of objects with a minimum bounding rectangle. Each node in the R-tree contains a rectangle, consisting of objects or multiple rectangles depending on the height of the node in the tree (Figure 2.1). The R-tree is adequate when requesting data from a finite historical movement dataset. Updates in the data, which are quite common when dealing with moving objects, are costly operations since each time the data is updated, the R-tree index must be updated as well. Xia and Prabhakar [2003] states that therefore the R-tree should not be proposed for indexing moving objects. Also Meng and Chen [2011] argues that R-trees are not that capable, especially not when storing the derived trajectories from the positions of moving objects. With an R-tree is it according to Meng and Chen [2011] not possible to store three-dimensional trajectories.

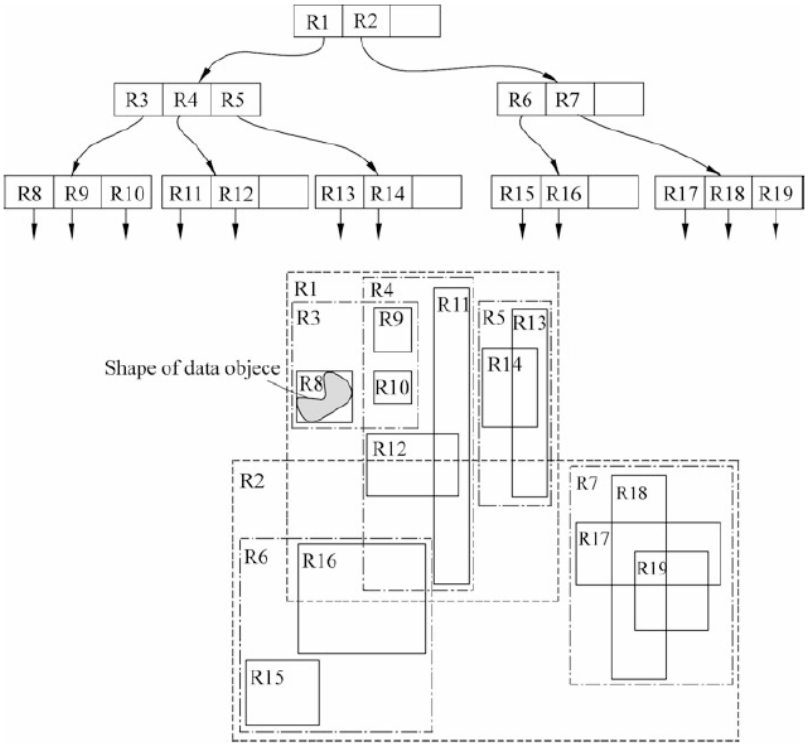

Figure 2.1: R-tree [Meng and Chen, 2011]

An R-tree variant which is able to store also trajectories in an efficient way is the Spatial-Temporal R-tree (STR-tree). A algorithm is used for finding the predecessor of each point. Each node then contains a identifier to this predecessor [Meng and Chen, 2011; Mokbel et al., 2003b].

For an efficient management of moving objects data worth mentioning is the 2+3 R-tree index method provided by Mokbel et al. [2003b]. This index aims to index both historic and current real-time position data at the same time. The 2+3 R-tree does this by using two separate R-trees. One R-tree is organized for the present positions and one R-tree is organized for the historical positions. Once the real-time position is updated this position is inserted into the R-tree for the historical positions and deleted from the present position R-tree.

Extending research exist on multi-dimensional spatial-temporal access methods [Mokbel et al., 2003b]Nguyen-Dinh et al. [2010]Meng and Chen [2011]. In practise though just a few multi-dimensional spatial access methods are implemented in the mainstream databases [van Oosterom, 1999]. This explains the extensive use of the R-tree in the storage of moving objects.

Indexing and clustering the multi-dimensional moving past one-dimensional

A Space Filling Curve (SFC) is a popular method for indexing and clustering moving objects. A SFC can be defined as a mapping of n-dimensional space into a one-dimensional curve. They deal with the fact that geospatial data has 2 or more dimensions and try to combine these into a single one-dimensional code. There are numerous kinds of SFC. Figure 2.2 shows a couple of them used in a two-dimensional space and figure **??** in three dimensional space. The difference between them is located in their way of mapping towards one-dimensional space [Mokbel et al., 2003a]. Two of the most use space filing curves are the Morton order curve and the Hilbert curve. Both are known to map multi-dimensional data into one-dimension while preserving the locality.

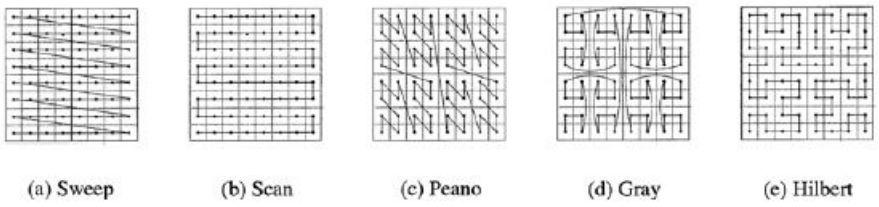

Figure 2.2: Various SFC 2D [Mokbel et al., 2003a]

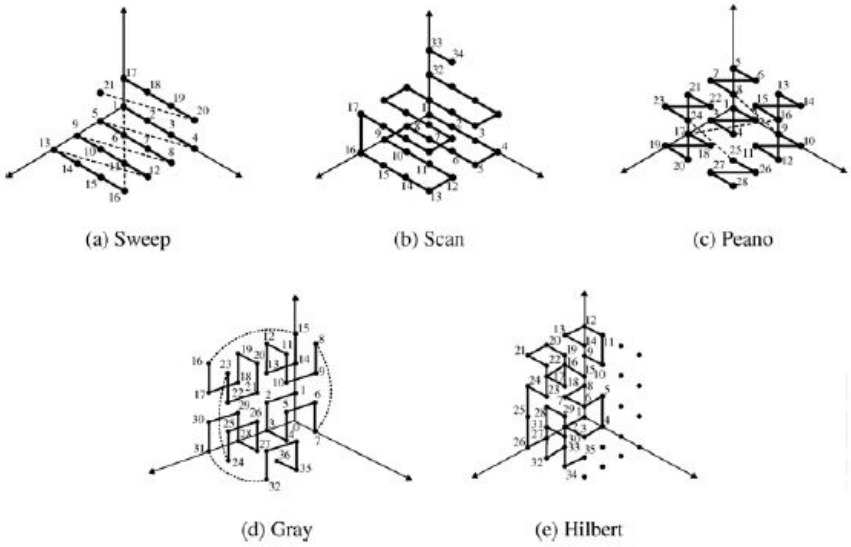

Figure 2.3: Various SFC 3D [Mokbel et al., 2003a]

A SFC will store the data, objects, that are close to each other in reality close to each other on disk. This clustering reduces the number of disk accesses and improves the response time.
Moon et al. [2001] and Lawder and King [2001] compare different SFC and

then particularly the Morton and the Hilbert curve. Concluding remarks from these studies are that the Hilbert curve deserves closer attention. The Morton order is mostly implemented in comparison with the Hilbert curve. Implementation and understanding of the Morton curve is less difficult.

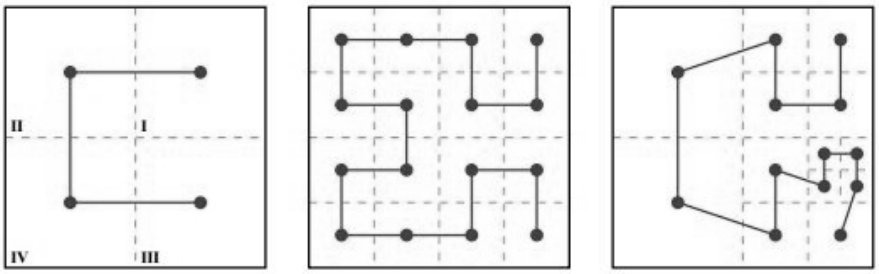

Figure 2.4: Hilbert SFC [Campbell et al., 2003]

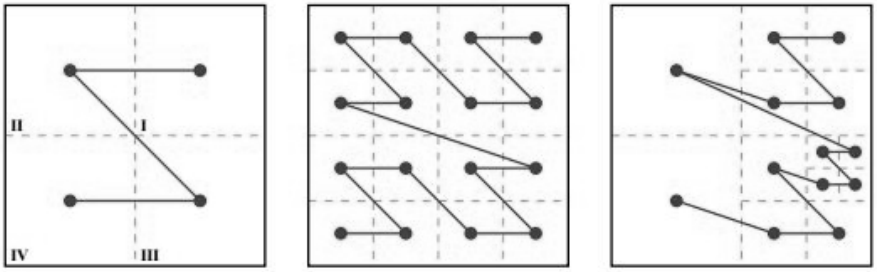

Figure 2.5: Morton order SFC [Campbell et al., 2003]

A big advantage of using a space filling curve while clustering moving objects is that each one dimensional data structure can be used for indexing the data Park et al. [2013].
B-trees are the indexing technique which is most used in combination with a SFC. The main characteristic of a B-tree is that it is designed to reduce the number of disc accesses. A B-tree is a tree consisting of three layers; root, internal and leaf nodes (Figure 2.6 visualizes only root and leaf nodes). Both the root node and the internal nodes consist of key-pointer pairs. This means that with the key they distinguish themselves from other nodes and with the pointer they are linked to their child nodes. The leaf nodes then have a pointer towards the data record.

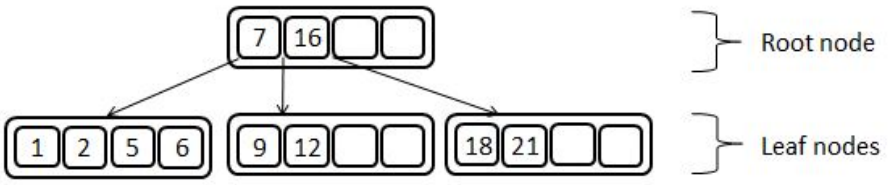

Figure 2.6: B-tree

Various research propose variants on this B-tree. Jensen et al. [2004] proposes the Bx-tree which does not store the individual points of a moving object but the linear function belonging to the movement of the object in

combination with the time this movement took place. This Bx-tree is according to [Chen et al., 2008a] the first effort in adapting the B-tree to index moving objects. Chen et al. [2008a] itself proposes the ST2B-tree, a tree which subdivides the regular B-tree into two B-tree's each assigned a time interval. When an object updates its position it will be indexed in one of the B-trees, based on in which time interval it fits. Whenever an object updates its position again this position then will be indexed in the other B-tree.

2.4 INDEX BENCHMARKING

Benchmarks can be defined as efficient methods to compare the performance of different databases for specified use cases. These comparison techniques can also be used to compare different indexes on a data set to conclude on the most efficient way to store and query this data set. Indexes are an important part in efficient data management. Evaluating the use of the index therefore is therefore a logical step. Chen et al. [2008a] and Duntgen et al. [2009] provide indexing benchmarks for the indexing of moving objects data. Duntgen et al. [2009] considers the efficiency of the execution of specified queries as the indicator if an index is useful. Different kind of queries are to be used in order to create a full understanding on how this index performs. Queries as known or unknown objects identity, spatial-temporal dimensions, query interval, condition type and aggregation will cover the possible types of queries than are asked about moving objects. Chen et al. [2008a] indicates an indexing benchmark as a tool to extensively evaluate the aspects of a moving objects index. Four metrics are the aspects on which the index is evaluated; the number of I/O's, the CPU time, the size of an index on disk and the response time of the query execution.

2.4.1 Querying moving objects

Queries are an important part within index benchmarking. There exist many types of queries that can be asked about moving objects. For example it is interesting when, where and how movements changed [Guting and Schneider, 2005]. Meng and Chen [2011] introduces some 'basic' querying types. The point query, range query, the nearest neighbour query and the density query are defined as the most common 'basic' queries [Meng and Chen, 2011; Lin, 2007].
A point query retrieves a point, a moving object, at a certain time or a certain location. A range query on a moving object database retrieves all objects within a certain geographical area (rectangle) during a time interval. This query combines time with spatial. A nearest neighbour query retrieves all objects nearest to the query position at a certain time. A density query retrieves the regions with a high density at a certain time.
Pfoser et al. [2000] cites that the nature of the data and the type of queries are of importance when designing an index method. The effectiveness of an index relies on how efficient the query is executed using this index. Index and cluster methods should keep those queries in mind.

3 METHODOLOGY

This thesis aims to find a way to efficiently store AIS data to support spatial-temporal data analysis. This chapter will provide and discuss the approaches and methods used to reach this aim.

A database that fits all cases does not exist. The right database is use case dependent (section 2.2.1) [Hecht and Jablonski, 2011; Klein et al., 2009; Lourencco et al., 2015]. Finding the right database to manage; store and structure AIS data is dependent on the set requirements, especially the requirements that are based on use cases.
Klein et al. [2015] provides a method for finding the right database which will be used in this research as a guideline not only to find a suitable database but also an effective and efficient database organization. This makes this method applicable for all use cases in which a database must be chosen and organized.
The method is divided in five stages: requirement specification, candidate database selection, data model design, performance and scalability tests and a report of the results (Figure 3.1).

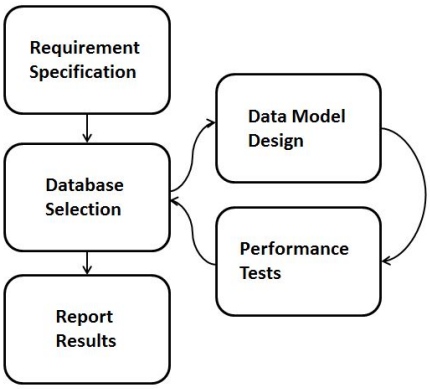

Figure 3.1: Database and Database Organization methodology, based on Klein et al. [2015]

3.1 CHOOSING A PROPER DATABASE

Where [Klein et al., 2015] uses these stages to find the 'perfect' database for a use case, in this MSc thesis only the first two stages are used to find a suitable database for AIS data management (chapter 4). The other stages are used to structure and optimize the chosen database for the handling of AIS data (chapters 5 and 6) The next subsections will describe the first two stages in more detail and additional methods used within those stages are explained thoroughly.

3.1.1 Requirement specification

The goal of the specification of the requirements is to indicate the AIS data features, determine the task the database has to do, in what way the data will be used and determine what specifications the database should have.

The features of the data that will be inserted into the database are most important within the selection of a proper database to manage, store and structure this data in. Data features as real-time, geospatial, and very large volume, can be the reason a certain database is not suited to store AIS data.

[Klein et al., 2015] explains that besides the data features, stakeholders are crucial in this stage, they define the database tasks and specifications. The tasks the database must fulfil in this MSc research will be specified by its stakeholders, the colleagues of the Centrale Informatie Voorziening (CIV) department of Rijkswaterstaat. They will design use cases for which they want to use the database containing historical AIS data for. Requirements the database than must fulfil are based on these use cases. Examples are the necessity for fast reads and look-ups and or continuous reads.

3.1.2 Database selection

The aim of this stage is to find a well suited database for the management of AIS data based on the in the previous stage set requirements. Santos et al. [2015] states that selecting a database depends on how well this database handles the use case specific data and how well it performs the most usual functions and queries the database is expected to execute. The selection of a database thus depends on the combination of the requirements based on the data features and the requirements based on the use cases.

The database selection in this MSc thesis will be made by comparing the specified requirements with database documentation and related literature. The database that in theory should manage AIS data will be chosen for the management of AIS data.

3.2 DATABASE ORGANIZATION

The following three stages from the on Klein et al. [2015] based method are used in the matter of structuring, organizing and optimizing the by theory chosen database efficiently.

3.2.1 Data modelling

Certain databases, like NoSQL databases, might not force one to create a schema, a model, for the data before insertion into the database. How data fits together, how data is related is still important though, especially when querying it. Asking questions to a database is directly proportional to the data structure. Knowing where the information that is to be queried is situated within the data is important for the development of the data(base) organization.

Modelling the data, the existing relations within the data is effectively done by creating a model with a **Unified Modelling Language (UML)**. [1] With the use of UML it is possible to model the AIS data (relationships) independently from a database type.

3.2.2 Data clustering and indexing

Retrieving fast output from queries makes cluster and index techniques for smart storage of the data necessary. Different cluster and indexing techniques exist to provide a close storage on disk for related data and a fast retrieval of this data (section 2.3).
The indexing technique used to locate vessel positions that in reality are close to each other close, will be first and foremost an on AIS data adjusted Morton order SFC (Figure 2.5). The Morton order SFC is known to be often applied to multi-dimensional point data. Indexing or clustering with the Morton order SFC is a logical first step in the conclusion if such a SFC is efficient in the storage and organization of AIS data. For the Morton SFC though applies the fact that points that are adjacent according to the curve not always are close to each other in reality. The Hilbert curve does not have this unfortunate property. Inserting a Hilbert SFC seems more efficient though more difficult to implement (Figure 2.4). But if a Morton order SFC can not organize the AIS data in an efficient, effective, manner there will be no guarantee that a Hilbert SFC will. The implementation of the Morton order SFC thus will be seen in this MSc as a 'proof of concept'.

The Morton order SFC requires an one-dimensional index. The index that will be used for this is the well known B-tree. This indexing tree is widely implemented in main stream databases and even though several researches present an improved version of the B-tree [Chen et al., 2008b; Jensen et al., 2004], it is a well working index structure for indexing the Morton codes.

3.3 DATABASE PERFORMANCE TESTS

Where database bench-markings are used to define the best performing database, index bench-markings will evaluate the performance of indexes. Several index bench markings researches exist, also bench-marks for the evaluation of moving object indexes Chen et al. [2008a], Duntgen et al. [2009]. The method used in this MSc thesis to evaluate the performance of the database organization and indexes will be based on the methods used by [Duntgen et al., 2009] and or [Chen et al., 2008a].
Where [Duntgen et al., 2009] locate the focus in the benchmark on an interesting range of queries, [Chen et al., 2008a] focusses on four metrics; the number of **I/Os** [2], the Central Processing Unit (CPU), the size of the index on disk and the response time.
To indicate the performance of the database organization the focus will be on the response time of specified queries, the size of the index and database on disk and the effectiveness of the used index (how many database records are searched). A combination of these four metrics will conclude on the efficient data(base) organization of AIS data.

1 **UML** is a widely used standard modelling language
2 **I/O** stands for Inputs and Outputs, it is related to the communication between subsystems of the computer

4 IMPLEMENTATION: DATABASE SELECTION

This chapter is the first of three implementation chapters, each containing their own stages of the proposed methodology based on [Klein et al., 2015]. This chapter will explain the selection of the database that will be used for storing the historical AIS data. The first section, section 4.1, defines the by Rijkswaterstaat and the AIS data itself set requirements which the database should meet. The second and last section, section 4.2 then explains the database selection in which the set requirements play the major role.

4.1 DATABASE REQUIREMENT SPECIFICATION

When finding a database suited for a specific use case, in this case the management of AIS data, the requirements a database must fulfil are leading. The requirements the database for the management of AIS data must accomplish depends on the features of the data and the use cases that are specified by colleagues of the CIV department of Rijkswaterstaat.
The first part of this section will clarify how the historical AIS database will be used and what features are necessary to accomplish this use, by describing use cases and the second part shall specify the data features and the related requirements.

4.1.1 Use cases

Rijkswaterstaat has the ambition to apply different spatial-temporal and technical analyses with the AIS data that the historical database will hold when developed. The different spatial-temporal analyses are briefly explained by use cases. Multiple different use cases were specified during personal sessions. The three main use cases, the use cases that all stakeholders, colleagues of the CIV department of Rijkswaterstaat appointed, will be used in this MSc thesis; Location, Trajectory, Bounding Box.

Location

The first use case is one concerning the location of a vessel. The most basic information to get from historical AIS data is where was a certain vessel at a specific time. This question is the base of every other spatial-temporal question concerning AIS data and therefore is most important.

Trajectory

The second use case concerns the extraction from AIS data to get the historical trajectories of vessels. These trajectories can be used in several applications since they answer multiple questions like; Where has a vessel been the last year? or What routes does this vessel normally take. One of these applications is route prediction, the prediction of future positions. Based

on previous taken routes it is possible to predict the route the vessel will take next. Knowledge about the destination and the current position of the vessel is then helpful.

Bounding Box

The next use case concerns a geographical area, namely the amount and location of the unique vessels that are located within a certain geographical area at a specified time. The amount of vessels inside a specified area can give knowledge about the density of specific parts of the inland waterways at specific times. This density information can be used to improve the traffic management.

4.1.2 Data features

Section 1.1 already explains shortly the different features of AIS data. This section will collect the ones which influence the selection of the database. AIS data can be defined as a combination of messages that vessels sent towards other vessels and to base stations. Table 4.1 gives an overview of the different messages with a short explanation of what kind of data these messages hold. These messages, 27 different kinds, contain unique and similar information. The 27 messages all have their own schema, their own information, their own values. Each message type on its own contains the same values, but due to the fact that there are 27 different ones and the update rate for each one is different and dynamic, the data will not be considered structured.
AIS data is real-time data. This real-time nature influences the data volume, it grows every second. Also the historical AIS data. Historical AIS data must be kept up to date. To what extent the historical data is to be up-to-date has to be decided by Rijkswaterstaat. This decision influences the update, insertion, rate of the data in the database. This MSc research does not test a real-time update stream of historical AIS data but keeps the necessity for handling fast updates, inserts, writes in mind for the selection of a database. Besides the update rate also the volume, which becomes bigger and bigger is an important feature of AIS data. Where one month of AIS messages already has a size around 100 Gigabytes, the size of several years will become Petabytes.

4.1.3 Requirements

The ambition of Rijkswaterstaat to conduct spatial-temporal analyses according to the three defined use cases, weights heavily on the necessity of fast extraction of information from the database: fast reads. To conduct up-to-date reads from the database, writes that update the data are to be performed before the reads.
Another important issue related to the performance of spatial-temporal analyses is that the database should be able to store and perform operations with (geo)spatial data with the use of complex queries.
Furthermore the selected database should be able to handle data that is not considered structured with a possible real-time update rate and a volume exceeding internal memory without losing the abilities of fast reads and writes.

ID	Message	Explanation
1	Position report	Scheduled position report; (Class A shipborne mobile equipment)
2	Position report	Assigned scheduled position report; (Class A shipborne mobile equipment)
3	Position report	Special position report, response to interrogation; (Class A shipborne mobile equipment)
4	Base station report	Position, UTC, date and current slot number of base station
5	Static and voyage related data	Scheduled static and voyage related vessel data report; (Class A shipborne mobile equipment)
6	Binary addressed message	Binary data for addressed communication
7	Binary acknowledgement	Acknowledgement of received addressed binary data
8	Binary broadcast message	Binary data for broadcast communication
9	Standard SAR aircraft position report	Position report for airborne stations involved in SAR operations, only
10	UTC/date inquiry	Request UTC and date
11	UTC/date response	Current UTC and date if available
12	Addressed safety related message	Safety related data for addressed communication
13	Safety related acknowledgement	Acknowledgement of received addressed safety related message
14	Safety related broadcast message	Safety related data for broadcast communication
15	Interrogation	Request for a specific message type (can result in multiple responses from one or several stations)(4)
16	Assignment mode command	Assignment of a specific report behaviour by competent authority using a Base station
17	DGNSS broadcast binary message	DGNSS corrections provided by a base station
18	Standard Class B equipment position report	Standard position report for Class B shipborne mobile equipment to be used instead of Messages 1, 2, 3(8)
19	Extended Class B equipment position report	Extended position report for class B shipborne mobile equipment; contains additional
20	Data link management message	Reserve slots for Base station(s)
21	Aids-to-navigation report	Position and status report for aids-to-navigation
22	Channel management(6)	Management of channels and transceiver modes by a Base station
23	Group assignment command	Assignment of a specific report behaviour by competent authority using a Base station
24	Static data report	Additional data assigned to an MMSI
25	Single slot binary message	Short unscheduled binary data transmission (Broadcast or addressed)
26	Multiple slot binary message	Scheduled binary data transmission (Broadcast or addressed)
27	Position report for long range Applications	Position report for long range Applications

Figure 4.1: List of the 27 different AIS messages

4.2 DATABASE SELECTION

The question that will be answered in this section is what (kind of) database should manage AIS data.

4.2.1 NoSQL

Traditional Relational databases are designed to manage relative static and structured datasets that fit into predefined tables [Hecht and Jablonski, 2011]. Section 2.2 explains the necessary features for storing moving objects and concludes that traditional relational databases might not be the right choice to store them. A NoSQL database therefore can be seen as a proper choice since they are to be the solution to the limitations of traditional relational databases.

The management of historic AIS data asks for fast reads from the database while using complex queries. The query possibilities of traditional relational databases are comprehensive. The Structured Query Language (SQL) used in these databases make the execution of these queries possible. Fast reads though are dependent on multiple features besides the ability of executing complex queries. They are dependent on the data, data volume, structure, cluster and indexes.
Management of historical AIS data might lead to the thought of the management of a finite data set, but the historical AIS data needs to be kept up to date. The term a finite data set therefore does in reality not apply to historical AIS data. Trying to store AIS data which is coming from an infinite data source within a traditional relational SQL database will result in problems concerning the update rate of the data and scalability problems as explained by [Moniruzzaman and Hossain, 2013] in section 2.2.
Scaling with a traditional relational database is only possible by adding more storage space to a computer or just running the database on a more powerful computer (vertical scaling) Van der Veen et al. [2012]. The technology providing these storage option has eventually a limit, there are only so powerful computers. Horizontal scaling means adding additional servers and distribute the existing data among these servers. Due to the complex relations this database can contain it is difficult to store one part of the data elsewhere. The ways relational databases use to improve their performance relies on the fact that a complete picture of the data is available. These performance improvements does not work if half of the data is stored on another server [Plugge et al., 2015].

NoSQL databases where thought of to be the solution for the limitations of traditional relational databases [Abramova et al., 2014]. The main features of NoSQL databases are that they do not use a relational model. Unstructured data can be stored easily since the database is schemeless [Indrawan-Santiago, 2012]. NoSQL databases are furthermore designed for horizontal scaling [Abramova et al., 2014]. These features already indicate that a NoSQL database should theoretically be able to manage the large volume of historical AIS data. This horizontal scalability affects the readability of the database when storing large volumes of data. A disadvantage of NoSQL databases is the fact that they provide less 'tools'. Because they do not make use of the SQL the query complexity will linger on traditional relational databases.

A last important aspect is that AIS data is (geo)spatial-temporal data. The database must be able to manage, store and perform operations with spatial data. Not many NoSQL databases are able to do this [Baas, 2012]. Spatial and or temporal features are widely used by traditional relational databases.

4.2.2 Document store

There exist four types of NoSQL databases; document store, graph based, key-value and column family databases. These four types of databases are different in their functionalities and each is specialized in certain use cases [Hecht and Jablonski, 2011].

Key-value store

Key-value databases have a simple data model containing keys with associated values which are only retrievable if the key is known [Amirian et al., 2013]. According to Han et al. [2011], key value stores favour high scalability over consistency. This means that key-values stores are more capable of handling a growing dataset compared to handling the requirement that each specified database task has to be performed. Query and analysis functions are therefore often left out of the key-value store capabilities.
The key-value pairs within AIS data are related, within the key-value store these key-value pairs are independent of each other. Relationships are possible to insert but only on application level [Hecht and Jablonski, 2011].

Graph database

Within a Graph database the relationships that connects the data are leading. They are efficient in the handling of linked data. Graph databases are best suited for data that can be modelled as networks [Amirian et al., 2013]. The AIS messages are primary linked by Maritime Mobile Service Identity-number (MMSI), each message sent by the same vessel or base station contains an associated MMSI. The messages from one vessel or base station, associated to one MMSI, which together form a trajectory, should be possible to store as a network. The MMSI is not the only feature that relate messages to each other. This makes the storage of AIS data within a network difficult, but not impossible. Though it is concluded that AIS data should not be stored within a graph database.

Column family database

Column family databases are systems that are able to store a large amount of different values within one row. The format of this database type is tabular like traditional relational databases. The inserted rows do not have to be constant to each other though. This way semi-structured data can be stored. The focus within column store databases lies as the name says within the columns. All data within one column is grouped together (as a family). Because of this the column family databases can compress the data highly. A row, the attributes of a AIS message when stored in a column family database, will be stored in different column families. This means that not all parts of the AIS message have to be extracted when asking for the value of one attribute. One critical note is that each value is stored on its own, although it is possible to store with each value the associated number of the row and column.

Another specification of column store databases is that they are designed to scale to very large sizes. Also they can handle a high rate of data updates.

Document store database

Document oriented databases make use of the key-value pairs concept but when compared to key-value stores the difference lies in the fact that the values and the keys can be queried [Indrawan-Santiago, 2012]). A document database exists of one or more collections (data sets) containing one ore more documents (data records) which contain one or more key-value pairs (attributes). Document store databases are mainly used due to their flexibility in data scheme (schemeless), their ability to handle complex data and their performance and scalability [Abramova et al., 2014; Sullivan, 2015]. Multiple attributes are easily implemented in document and documents can easily be filtered based on these attributes. These functionalities make document store databases a suitable database kind to store AIS data.

4.2.3 MongoDB

As can be concluded from section 4.2.2 document store based NoSQL databases should be able to manage, store and structure AIS data. From this database types there is only a select group of different databases that have added geospatial functionalities.

The database that should be used for the management of AIS data is MongoDB. This is a popular open source NoSQL document oriented database of which the developers know that it wont be a fit for all uses. MongoDB is according to Plugge et al. [2015] aware that it is not suitable for each use case, and the database is not designed for that either.
MongoDB is developed with the knowledge that there are a great deal of features of traditional relational databases that work good. Such as indexes and complex queries. These features therefore will work the same in MongoDB as they should work in traditional relational databases. MongoDB has an expressive query language and the possibility to implement secondary indexes. Other main features of MongoDB are strong consistency, a flexible data model, horizontal scalability and high performance. MongoDB is therefore usable with the storage of complex data and performing analysis [MongoDB, 2016]. With this features MongoDB meets the set requirements (section 4.1).

MongoDB stores data as documents. Internally these documents are stored as Binary JSON (BSON) which compresses the data extensively. The data in the documents consists of fields in combination with a value (key-value pairs). The documents are stored into what is called a collection. Within a collection it is possible to store documents which have similar structures but this does not have to be the case. A collection can be defined as a selection of different documents. Documents within a collection do not have to have the same amount of key-value pairs within them (schemeless)(Figure 4.2). This feature is important when storing AIS data, it means that it is possible to store all 27 different AIS messages (which contain of different amounts of key-value pairs) together in one collection.

Querying stored data in MongoDB can be defined as finding specific fields with specific values within the documents of a collection. It is pos-

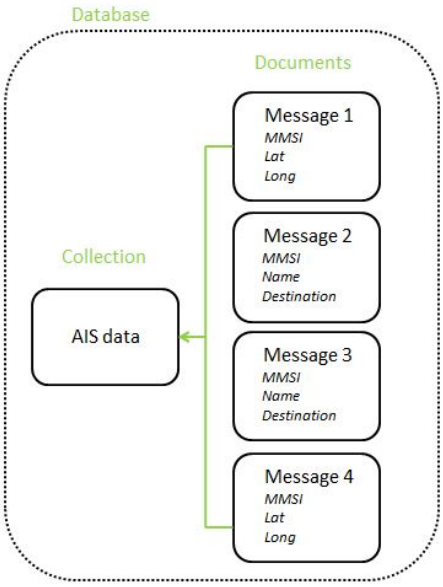

Figure 4.2: MongoDB; Collection and Documents

sible for these queries to become complex. MongoDB is able to handle queries comparable to SQL queries, defined in the programming language from which MongoDB is controlled in combination with the query language of MongoDB itself. MongoDB is controllable by different programming languages, it contains different drivers connected to different languages [MongoDB, 2016].
MongoDB also offers a tool for interaction with the database independent of the drivers; the mongo shell (Figure **??**). This is an interactive JavaScript shell in which the operations that can be performed with the use of drivers can be executed as well.
Finding the specific fields and values specified within the queries is primarily done by searching all documents within a collection (search without any index). This feature makes horizontal scalability possible. When the collection of documents is spread among different servers, each server checks its documents on the specified keys and values and returns a result.
A quick look up is arranged by the implementation of indexes. Indexes in MongoDB can be declared on any field in a document. It is possible to implement an index on each field within a document.
When querying documents based on two or more fields which all contain an index, MongoDB performs query optimization. This means that it will select the index that executes the query the most optimal way. Choosing which index is to be used is also possible.

5 IMPLEMENTATION: DATABASE ORGANIZATION

This chapter is the second of the three implementation chapters. This chapter will explain the process of organizing the selected database MongoDB. This process of organizing the database is a central point in this research. A well arranged database organization is the base to a fast retrieval of data. The first section of this chapter, section 5.1 will explain AIS data and its construction. This section will conclude with the way in which the data should be organized into the database to support the specified use cases and or spatial-temporal analyses. Section 5.2 then explains the processes of necessary data transformations to accomplish the in section 5.1 specified organizations. The chapter then will end with the explanation on how the AIS data is loaded into MongoDB.

5.1 DATA MODEL

Even though NoSQL databases do not need a specified data model or scheme, it is still interesting and useful to look at the relations existing within the AIS data. These relations clarify the construction of the AIS data. Knowledge on the construction of the data provides information on where the necessary data is located.
To support the spatial-temporal analysis by Rijkswaterstaat specific information which is concealed in the AIS data is necessary. In order to extract this information an overview of the structure and the relations between the data attributes is necessary to get knowledge on where to find this required information.

5.1.1 AIS messages relations

The relations between the 27 existing different AIS messages are visualized in Figure 5.1. The central class is the AIS message. All different messages are connected to this class via inheritance. The attributes of the central class are the **MMSI** [1], the repeat indicator and the date-time (Figure 5.2). All AIS messages contain these three attributes.

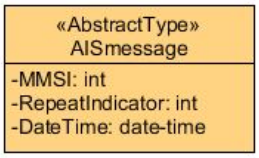

Figure 5.2: AIS message Central class

1 The **MMSI** is the number of the vessel, which is unique for each vessel

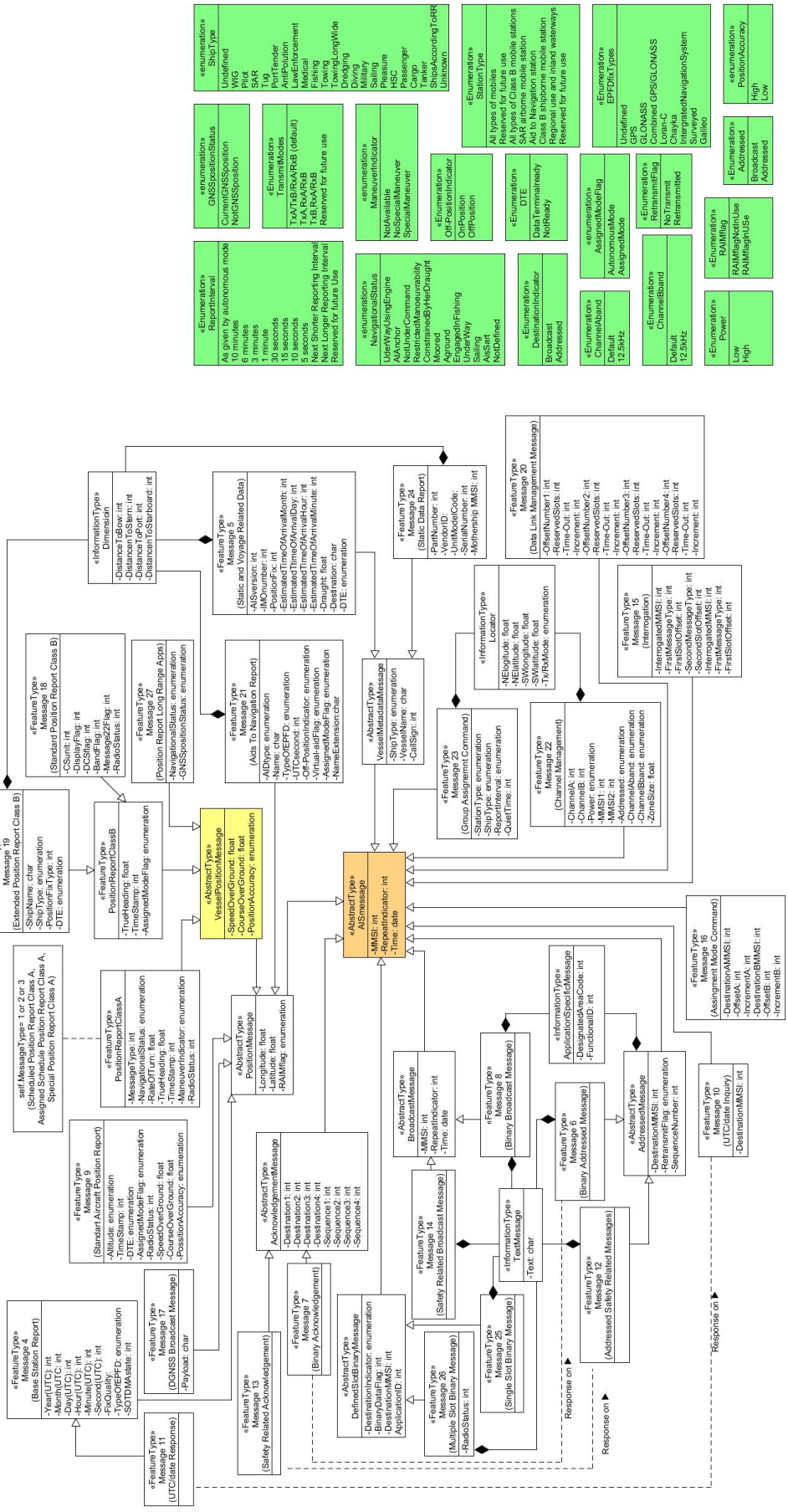

Figure 5.1: The UML diagram of AIS data

This central AIS message is related to 6 abstract classes (Figure 5.3) which in there case relate to 21 of the 27 different messages. The AIS message can be divided into 6 different main message types (abstract classes): Position messages, metadata messages, acknowledgement messages, defined slot binary messages, addressed messages and broadcast messages. The position messages then can be specified by two different kind of position messages: vessel position messages and other position messages (Figure 5.3).

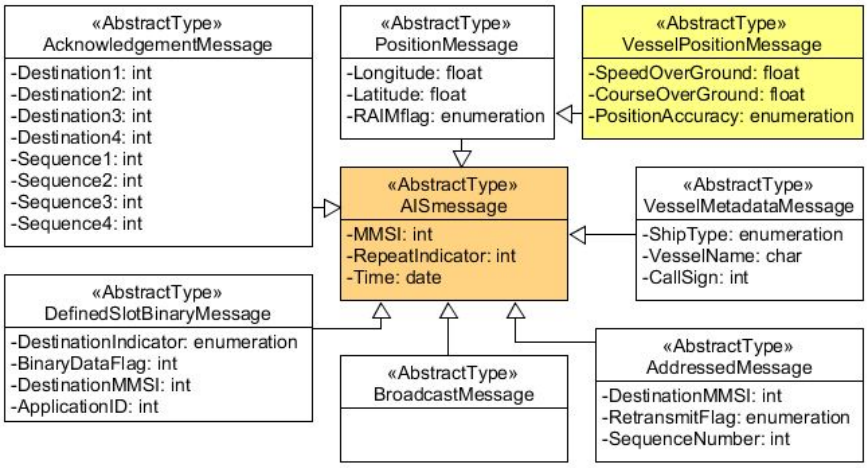

Figure 5.3: Abstract classes

Six different messages are related to the Vessel Position class, six messages contain a position of a vessel (Figure 5.4). Two different messages are related to the Metadata class, this means that two messages contain the additional information of a vessel such as name and destination. Two messages can be defined as acknowledgement messages, they are response messages that let the vessel or base station sending addressed messages know that these were received. Binary and broadcast messages are subdivided into 4 messages. The messages directly related to the central message class are messages that are sent by base stations to vessels to adjust the way their messages are sent.

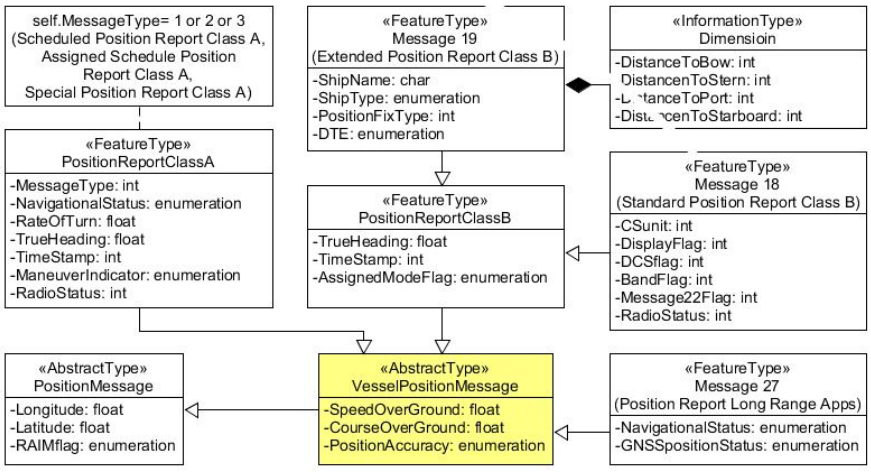

Figure 5.4: Vessel position messages

Two data(base) organizations

Taking a close look at the data model and the use cases together it becomes clear that only a few messages contain the necessary information for answering the specified use cases. And from these messages only some attributes are useful, others are not required.
To answer both the *location* use case and the *bounding box* use case four attributes are necessary: the latitude, longitude of a vessel position, the MMSI and the date-time. To answer the *trajectory* use case only three of those attributes are necessary: the latitude, the longitude and the MMSI
The four different attributes containing the information significant for answering the use cases are all located within the abstract class VesselPositionMessages (Figure 5.1). Six of the 27 messages contain the necessary information. The other messages are factual not needed.
To not throw away data when only storing the four previous mentioned attributes, the original encrypted message which hold all information should be stored as well besides the attributes. From the six messages that contain the necessary attributes, these extracted attributes in combination with the original AIS message should be stored. From all other messages only the storage of the original encoded message will suffice.

The three use cases used in this MSc research are not the only use cases appointed by Rijkswaterstaat. They give an indication of the bigger picture concerning spatial-temporal analyses. When organizing the data(base) focussed only on fast execution of the three specified use cases; location, trajectory and bounding box, the support for spatial-temporal analyses might not be that effective when other attributes are asked.
The storage of the original AIS message together with the extracted attributes for fast data extraction to answer the use cases, can also enable spatial-temporal analyses. Though only when this message is decoded after retrieval from the database.

To really support spatial-temporal analyses, another data(base) organization is proposed, one able to handle all possible spatial-temporal use cases without additional decoding processes. To support spatial-temporal analyses all attributes existing within the AIS data should be extracted and stored.
With the future in mind, to support not only the specified use cases but further spatial-temporal analyses, this division in two database organizations, **use case focussed and focussed on spatial-temporal analyses**, seemed imminent to find a way to manage, store and structure AIS data.

5.2 DATA PRE-PROCESSING

In order for AIS messages, to be readable, usable and storable conversion is necessary. The extraction of the attributes is not possible when the original AIS message is not decoded. AIS data is NMEA-0183 encrypted when it is received. These encrypted lines of data ask for decoding and reformatting pursuits before implementation in the specified data organization into MongoDB is possible. Section 5.1.1 concluded on the comparison between two database organizations, this means two different data preparation processes. What remains the same is the decoding process and the loading into the database (Figure 5.5).

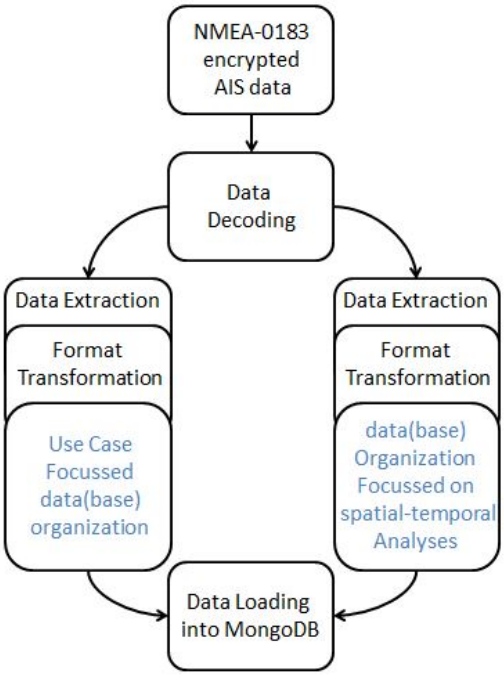

Figure 5.5: Data preprocessing work flow

5.2.1 Decoding AIS

A received AIS message contains two parts of data. The first part of the message is the date-time. This date-time is the date and the time on which the message was received not to be confused with the time the message was sent. The second part is the NMEA-0183 encrypted sentence and contains all valuable information apart from the date-time (Figure 5.6). Decoding this message is necessary to obtain the required information.

```
'2015 -11-01T00:00:07Z'  !ABVDM,1,1,7,A,13'fl1PP000KuM'MGEDPO?v@085@,0*5F
```

Figure 5.6: Date-time and NMEA-0183 Encrypted AIS message

To decode the lines into a readable format a decoding algorithm is used. This algorithm, written in the python programming language, reads the encoded messages line per line and for each line decodes the message and writes an output file with the decoded information (Figure 5.7). The algorithm uses a library specified to decode AIS messages developed by [Schwehr, 2015] which is only usable in a Linux environment. The decoding of the AIS data done by Martijn Meijers and Wilko Quak resulted in a file containing one month of decoded AIS data.

```
'2015 -11-01T00:00:07Z', 'slot_timeout':6, 'sync_state': 0, 'true_heading': 511, 'sog': 0.0, 'rot': -
731.386474609375, 'nav_status': 0, 'repeat_indicator': 0, 'raim': True, 'id': 1, 'slot_number': 647,
'spare': 0, 'cog': 296.79987792968, 'timestamp':  18, 'y': 53.00119400024414, 'x': 5.708691596984,
'position_accuracy': 1, 'rot_over_range': True, 'mmsi': 244630031, 'special_maoeuvre': 0
```

Figure 5.7: Decoded AIS message; Key-Value pairs

5.2.2 Data preparation for the Use Case focussed data(base) organization

The use case focussed data organization asks for the storage of only the MMSI, latitude, longitude and time. Data wastage is not an option therefore the "unnecessary" data will be stored by adding the original AIS message. From the data model (figure 5.1) can be derived the fact that time and MMSI is present in all AIS messages and the two other metrics; latitude, longitude can be found within the Vessel Position messages. Only the MMSI does not say anything about a vessel. Some metadata should be provided, such as the name, destination, draught, type and cargo just to provide some more knowledge about the vessel. This information is found within the two Metadata messages. The Vessel Position messages and the Metadata messages are thus most important.
The importance of the two message types led to the division of the AIS data in three sections or in the terms of MongoDB; collections. One collection containing the Vessel Position messages, one collection containing the Metadata messages and one collection containing all other messages. The three parts of AAIS data can be inserted into one collection, but to make the database understandable and make retrieval of data from the database easy and fast three collections will be developed, one for each kind. The extraction of the necessary information from the decoded AIS data and the division into three collections is done automatically using an algorithm of which the following pseudo code 5.1 explains how. MongoDB works best with JSON formatted files therefore the algorithm writes the three collections of messages to three JSON files.

Algorithm 5.1: Data Pre-processing algorithm Use Case focussed data(base) organization

Input: Decoded AIS file
1 **if** *the AIS message IS a Vessel Position message (message 1, 2, 3, 18, 19 or 27)* **then**
2 Extract the MMSI, latitude, longitude, time and the original AIS message.
3 **Write for each message the extracted data to a JSON file, one message per line**
4 **if** *the AIS message IS a Metadata message (message 5 or 24)* **then**
5 Extract the MMSI, destination, name, draught, type, cargo and the original AIS message
6 **Write for each message the extracted data to a JSON file, one message per line**
7 **if** *the AIS message IS NOT a Vessel Position OR a Metadata message* **then**
8 Extract the MMSI and the original AIS message
9 **Write for each message the extracted data to a JSON file, one message per line**
Output: Three JSON files ready to be implemented into MongoDB

5.2.3 Data preparation for the data(base) organization focussed on Spatial-Temporal Analyses

A data(base) organization with the focus on the future, on spatial-temporal analyses needs a fast retrieval of all information that can be retrieved from

the AIS data. This lead to the assumption that all decoded AIS information should be stored within MongoDB as related key-value pairs.
As in the data preparation for the use case focussed data(base) organization (section 5.2.2) a division of the decoded AIS data will be made in message types. The AIS messages will be divided into three sections: Vessel Position messages, Metadata messages and all other messages.
The following pseudo code 5.2 will explain the used algorithm which extracts, reformat and writes the AIS messages to three different JSON files.

Algorithm 5.2: Data Pre-processing algorithm for the data(base) organization focussed on spatial-temporal analyses

Input: Decoded AIS file
1 **if** *the AIS message IS a Vessel Position message (message 1, 2, 3, 18, 19 or 27)* **then**
2 Extract all the associated attributes
3 **Write these extracted attributes to a JSON file, one message per line**
4 **if** *the AIS message IS a Metadata message (message 5 or 24)* **then**
5 Extract all the associated attributes
6 **Write these extracted attributes to a JSON file, one message per line**
7 **if** *the AIS message IS NOT a Vessel Position OR a Metadata message* **then**
8 Extract all the attributes
9 **Write the extracted attributes to a JSON file, one message per line**
Output: Three JSON files containing al AIS information, ready to be implemented into MongoDB

5.3 DATA LOADING

The data format transformation, data division and re-organization resulted in two different data(base) organizations each consisting of three JSON files which are to be imported into MongoDB.
Data loading in MongoDB is done by inserting a *mongoimport* command in the MongoShell. As is explained in section 4.2.3 this MongoShell is a tool to communicate with the database. Such a *mongoimport* command must contain the name of the database and the name of the collection a file is to be inserted in as is the directory to this file.
For each data(base) organization three of these commands are necessary to populate the database, one for each collection.
Each JSON file has on every line a AIS message. Where the whole JSON file is implemented into a collection is each AIS message transformed into a document in MongoDB. Figures 5.8, 5.9, 5.10, 5.11, 5.12, 5.13 visualize how for each data(base) organization the data (vessel positions, metadata and all other messages) is stored within the database when the JSON data is loaded in.

```
db.positions.findOne()

        "_id" : ObjectId("57c98a6842861dd63038280c"),
        "y" : "51.871704",
        "x" : "4.312366",
        "mmsi" : "211535300",
        "message" : " \t['!ABVDM,1,1,8,A,139g5i0P00PCgHpMcVCVdwwnR15n,0*66\\r\\n
]\n",
        "DateTime" : "2015-11-01T00:00:00Z "
```


Figure 5.8: Data(base) organization use case focussed, vessel positions inside MongoDB


```
db.positions.findOne()

        "_id" : ObjectId("57c67250dcdc4fa3db4bd3f9"),
        "slot_timeout" : "0L",
        "sync_state" : "0L",
        "mmsi" : "211535300L",
        "repeat_indicator" : "0L",
        "sog" : "0.0",
        "timestamp" : "59L",
        "nav_status" : "0L",
        "true_heading" : "511L",
        "raim" : "True",
        "DateTime" : "2015-11-01T00:00:00Z ",
        "rot_over_range" : "True",
        "spare" : "0L",
        "cog" : "171.5",
        "x" : "4.312366485595703",
        "y" : "51.87170410156251",
        "slot_offset" : "4470L",
        "position_accuracy" : "1L",
        "rot" : "-731.3864746093751",
        "id" : "1L",
        "special_manoeuvre" : "1L"
```


Figure 5.9: Data(base) organization spatial-temporal analyses focussed, vessel positions inside MongoDB


```
db.metadata.findOne()

        "_id" : ObjectId("57d14feada804de62b41d6b8"),
        "TypeAndCargo" : "30",
        "mmsi" : "244860699",
        "destination" : "u'-@@@@@@@@@@@@@@@@@@@'",
        "DateTime" : "2015-11-01T00:00:00Z",
        "message" : " \t['!BSVDM,2,1,2,B,53aQ7612E:oe09M0001TF377R0ptu8@h4p@0000
4H85440Ht0;@000000000,0*0A\\r\\n', '!BSVDM,2,2,2,B,00000000000,2*3C\\r\\n']\n",
        "name" : "u'YE 118 NOORDLAND@@@@'"
```


Figure 5.10: Data(base) organization use case focussed, metadata inside MongoDB


```
db.metadata.findOne()

        "_id" : ObjectId("57c5792af7b0ca15a62f5efa"),
        "fix_type" : "1L",
        "eta_hour" : "24L",
        "repeat_indicator" : "0L",
        "DateTime" : "2015-11-01T00:00:00Z ",
        "callsign" : "u'PBWP@@@'",
        "id" : "5L",
        "type_and_cargo" : "30L",
        "ais_version" : "1L",
        "destination" : "u'-@@@@@@@@@@@@@@@@@@@'",
        "mmsi" : "244860699L",
        "imo_num" : "9775995L",
        "draught" : "0.0",
        "dim_d" : "4L",
        "dim_b" : "8L",
        "dim_c" : "5L",
        "dim_a" : "35L",
        "eta_day" : "0L",
        "eta_minute" : "60L",
        "spare" : "0L",
        "name" : "u'YE 118 NOORDLAND@@@@'",
        "dte" : "0L",
        "eta_month" : "0L"
```


Figure 5.11: Data(base) organization spatial-temporal analyses focussed, metadata inside MongoDB

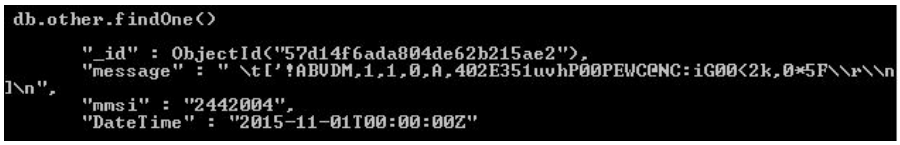

Figure 5.12: Data(base) organization use case focussed, all other messages inside MongoDB

Figure 5.13: Data(base) organization spatial-temporal analyses focussed, all other messages inside MongoDB

6 IMPLEMENTATION: DATABASE PERFORMANCE

This third and last implementation chapter will explain the process of optimizing and testing the index and data(base) organization performance. The first section, section 6.1, will explain the insertion of a 4D Morton order SFC index for an assumed better performance and how the AIS data is assumed to be clustered within MongoDB. Section 6.2.2 defines the testing of the performance of the indexes and data(base) organizations by two data volumes and three designed queries (section 6.2.3).

6.1 INDEXING AND CLUSTERING

The AIS data is stored per message, each message is one document belonging either to the vessel positions, metadata or the other messages collection as explained in section 5.1. The messages are stored either containing all attributes or containing only four attributes and the encrypted AIS message itself. The optimization method, the Morton curve, is yet to be implemented.

For the indexing, clustering, of AIS is a 4D Morton curve desired. The desire, the necessity for a 4D Morton code to manage AIS data lies within the three use cases which ask specifically for a 2D position, a date-time and a MMSI. Four attributes of which it will be efficient, for answering the use cases, if they are located close together.
Longitude and latitude (x and y) are to be stored close together so the locations of vessels that are geographically close to each other will be close on disk. This is necessary for the *location* use case as well as for the *bounding box* use case which ask for the location of one or more vessels. It is possible to apply a 2D Morton curve based on the 2D locations of vessels but because AIS data is moving objects data where time is equally important as location, a 2D Morton curve will not do.
Without a notion of the time and looking only at the data, a lot of vessels will be crashed into each other since they have sailed at the same location. When adding a notion of time per location it will become clear that a vessel will have been at the same place as another vessel but at a different moment in time. The importance of a 2D location in combination with time could evolve in a 3D Morton curve.
To substantiate the 4D Morton curve the focus is on the trajectory use case. This use case asks for all historic positions of one vessel. This question combines four attributes together, the 2D location, the time and the MMSI of the vessel. Locating the four attributes together then is efficient for the extraction of the trajectory of a vessel. Figure 6.1 visualizes the AIS data in 4D.

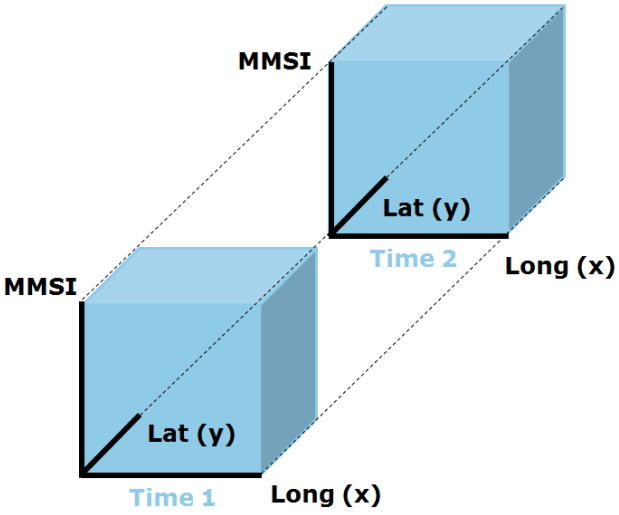

Figure 6.1: AIS 4D

6.1.1 A 4D Morton code index

There exist already several algorithms that calculate Morton codes. One of these algorithm is made available by [Psomadaki and Martinez, 2016] under the Apache Licence 2.0. This algorithm is able to decode and encode a Morton code for 2D, 3D and 4D data.
The calculation of a Morton code is done through **bit-interleaving**. [1] At first the set of four (latitude, longitude, MMSI and date-time) integer values (they have to be integers) are converted from decimal numbers to binary and then the values are rearranged. Figure 6.2 gives an example on how this calculation of a 4D Morton code takes place.

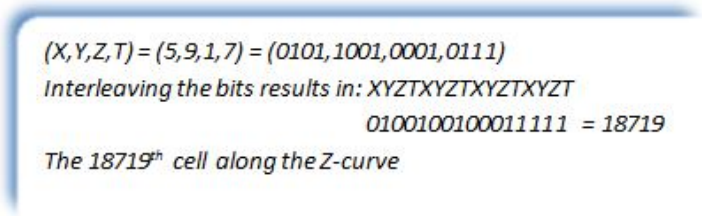

Figure 6.2: Calculating the Morton code (Example)

This example considers the space (X,Y,Z) and time (T) dimensions which are within this 4D point as equal, it is an integrated space time approach. When one of the dimensions for example time is treated as superior the Morton code will be different. Time will come first an then the three other dimensions. Instead of X-Y-Z-T-X-Y-Z-T etc. you will get a Morton code like this: T-T-T-T-X-Y-Z-X-Y-Z-X-Y-Z-X-Y-Z.
In this MSc thesis the integrated space time approach will be used, this means that the space, time and MMSI (id) dimensions will be treated equally, like they have the same dimensions. The fact that SFCs are based on hyper

1 Bit-interleaving is a method to make data retrieval more efficient by rearranging the data

cubes, which means that all dimensions in the curve should have a similar size, substantiates this. Scaling is necessary to to make this possible.

Scaling

Space and Time do not have the same semantics or nature. Space is measured in degrees, meters, or centimetres where time is measured in years, months, days, hours, minutes or even seconds. Scaling the two can be defined as the factor of how much time is integrated with how much space.

Vessels have speed limits on the (Dutch) waterways, just as when driving a car. Their speed limits though are combined with rules sets. For example, the speed limit for a vessel is 20 kilometres per hour when: they sail within 20 meters of shore, within 50 meters of swimming water, nearby games, parties and or demonstrations, within the harbour and between sundown and sunrise [Politie, 2016].
In order to comply with the speed of a vessel, space will be transformed into meters and time into seconds. The latitude and longitude of the position of a vessel will be conversed to meters by the transformation of the by AIS used coordinate system **WGS 84** [2] to the coordinate system of the Netherlands; (Amersfoort) RD new. The time, now in years, month, days, hours, minutes, seconds will be converted into **UNIX** [3] seconds. This means the time is measured in seconds from the 1st of January 1970 until now. The following algorithm will explain how these conversions took place in an automated way, for each vessel position message.

Algorithm 6.1: Space and Time conversions

Input: The latitude, longitude and date-time of each Vessel Position message

1 **foreach** *longitude and latitude* **do**
2 convert from WGS 84 to Amersfoort RD new
3 **foreach** *date-time* **do**
4 convert from year-month-day-hours-minutes-seconds to an UNIX time stamp in seconds

Output: The converted space (into meters) and time (into seconds) ready to be used for the Morton code calculation

A Morton code per vessel position

The 4D Morton code will only be calculated and used by vessel positions. They contain a latitude, longitude, MMSI and a time all together. Metadata messages do not contain a latitude or longitude, therefore calculating the 4D Morton code is not possible.
The data preparation of both data(base) organizations consists of several stages (section 5.2).
The stage in which the Morton code is calculated will be implemented between the extraction of the necessary data and storing this data in a JSON file (per message) stage. In this way the Morton code is added to the necessary data before it is stored in a JSON file and loaded into MongoDB (Figure 6.3) .

2 **WGS 84** is the reference coordinate system used by the GPS
3 **Unix** time is a system for describing instants in time defined as the number of seconds elapsed since 00:00:00 at the first of January 1970

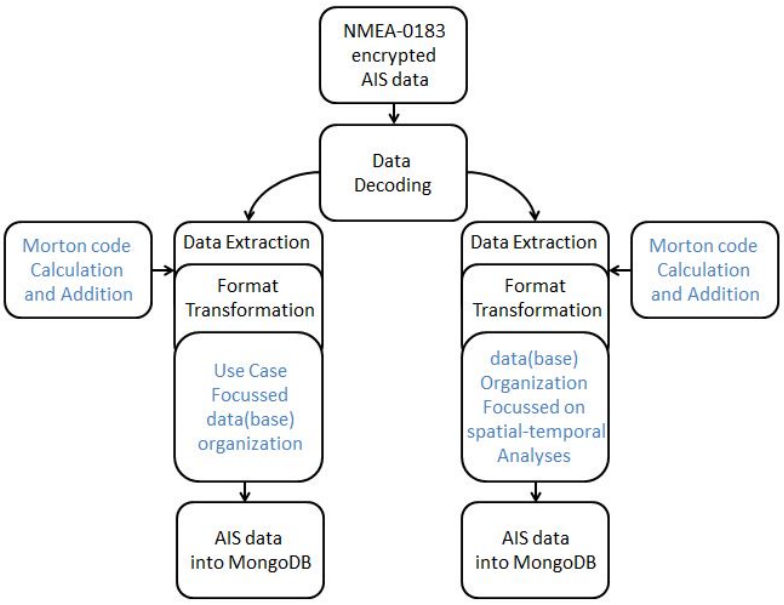

Figure 6.3: Morton code calculation and addition to the data

This arrangement changes the algorithms 5.1 and 5.2 into the next algorithm where the necessary data is extracted based on the used data(base) organization and the Morton code is calculated and added towards the message before the messages are stored into JSON files (algorithm 6.2)

Algorithm 6.2: Data Pre-processing algorithm with the calculation and insertion of the Morton code

Input: Decoded AIS file

1 **if** *the AIS message IS a Vessel Position message (message 1, 2, 3, 18, 19 or 27)* **then**

2 Extract the necessary data depending on the chosen data(base) organization (use case focussed or focussed on Spatial-Temporal analyses).

3 Transform for each message the latitude and longitude in Amersfoort RD new meters and the date-time in UNIX seconds.

4 Calculate the 4D Morton Code with the algorithm of Psomadaki and Martinez [2016] and add the calculated code to the extracted data. **Write the extracted data to a JSON file, one message per line**

5 **if** *the AIS message IS a Metadata message (message 5 or 24)* **then**

6 Extract the necessary data depending on the chosen data(base) organization (use case focussed or focussed on Spatial-Temporal analyses)

7 **Write the extracted data to a JSON file, one message per line**

8 **if** *the AIS message IS NOT a Vessel Position OR a Metadata message* **then**

9 Extract the necessary data depending on the chosen data(base) organization (use case focussed or focussed on Spatial-Temporal analyses)

10 **Write the extracted data to a JSON file, one message per line**

Output: Three JSON files ready to be implemented into MongoDB

Loading data in MongoDB with the Morton code as _id

Loading the data into the database will be at one part differently from the data loading as explained in section 5.3. The data that per message type is loaded into a MongoDB collection and per message is loaded into a Document consists of key-value pairs. The calculated 4D Morton code will be inserted into a message, a document, as one of these key-value pairs. The key name, the field, of the Morton code will be _id. But instead of having "morton" as a key and the code as the value, the key will be _id, the value is the Morton code (Figures 6.4, 6.5).

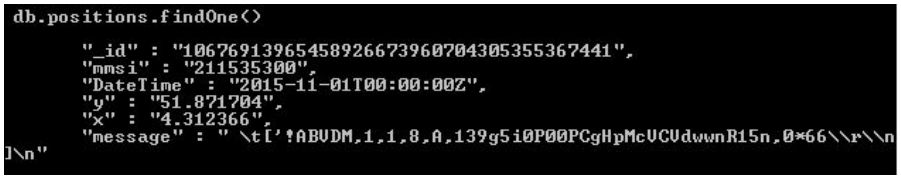

Figure 6.4: Data(base) organization use case focussed with Morton code inside MongoDB

Figure 6.5: Data(base) organization spatial-temporal analyses focussed with Morton code inside MongoDB

_id is in MongoDB used to mark each document that is inserted into the database with a unique Key(-Value). When inserting a collection of documents or one document to an existing collection into MongoDB, the database will automatically add a unique key(-Value) to each inserted document. It is possible to insert your own unique Key(-Value) instead of letting MongoDB do this automatically. The Key(-Value) must be a unique value for each document. Inserting the AIS data holding the 4D Morton code is done by letting the Morton code be _id, the unique Key(-Value) of the document (Figure 6.4 and 6.5).

The unique Key(-Values) then are automatically indexed by MongoDB with a B-tree, the data thus is indexed by a B-tree based on the Morton code.

6.1.2 Different indexes

A 4D Morton code index locates the vessels that have the same MMSI and are close to each other in space and time close together. Within none of the

three use cases all four attributes are known. The location use case searches for the location of a known vessel at a known time. The trajectory use case searches for all locations of a known vessel at a known timespan, and the bounding box use case searches for the vessels that are located within a certain bounding box at a certain time. The use cases therefore always ask for one or even two of the four attributes of which the Morton code exists.
Inserting an index in MongoDB is possible on each and all fields within a document. To substantiate the performance of the 4D Morton index it is decided to insert also a B-tree index on the four attributes individually.

6.1.3 Clustering

It was thought that the _id value was responsible for the clustering of the data, how the data is stored on disk, after implementation though it is assumed that the way data is stored on disk depends on how the data is inserted into the database. But no documentation exist on whether this is actually true.
Inserting the AIS data with or without the Morton code as _id is assumed to be automatically done based on date-time. The oldest AIS messages are inserted first. The AIS data therefore is clustered based on date-time.
A clustering on the Morton code is not enforced due to fact that the data was not sorted based on the Morton code after its calculation and implementation into the data. Inserting the AIS data holding this Morton code is thus done based on the date-time.

6.2 PERFORMANCE TESTING

Two data(base) organizations (two approaches) are developed and five different indexes are implemented. The aim to find an effective data management solution for AIS data, testing the performance of the developed solutions therefore is necessary.
Testing the performance is a comparison between the two database organization, and the five indexes individually or in combination with the database organization approach.

6.2.1 Data

The testing of the performance will be done with two different data volumes; a day of AIS messages and a week. A day of AIS data consist of 15.361.707 messages, and a week of AIS data consist of 83.223.293 messages.
Both for the day and for the week of data, the AIS data is pre-processed (section 5.2) and inserted (section 5.3) into the database. One database is organized with one day of data with the focus on the use cases with a Morton code index, One database is organized with one day of data with the focus on the use cases without a Morton code index, one database is organized with one day of data with the focus on spatial-temporal analyses with a Morton code index and one database is organized with one day of data with the focus on spatial-temporal analyses without a Morton code index. This results in 4 different databases with one day of data. This organization is also done with one week of data which results in a total of 8 different databases each containing three collections (Figure 6.1).

	Database Organization	Volume
Day	Use case focussed	1.517 GB
Week	Use case focussed	8.298 GB
Day	Use case focussed Morton Index	1.654 GB
Week	Use case focussed Morton Index	12.195 GB
Day	Spatial-temporal analyses focussed	12.619 GB
Week	Spatial-temporal analyses focussed	72.440 GB
Day	Spatial-temporal analyses focussed Morton Index	15.311 GB
Week	Spatial-temporal analyses focussed Morton Index	83.640 GB

Table 6.1: *Different data(base) organizations and their volume*

Each of those eight databases have a different volume. The one with the largest volume is the one containing one week of data with a index on the Morton code. This database has a size of 83.640 Gigabytes. MongoDB therefore has compressed the original size of decoded AIS data of a week of 300 Gigabytes with a factor 3. The size of the encoded AIS data though was 96 Gigabytes holding a month of data, which results in around 24 Gigabytes for one week. Comparing this size to the 83.640 Gigabytes of the database holding all decoded AIS data with a Morton code, the size has increased with 60 Gigabytes. Although, when just a few decoded AIS data attributes in combination with the encoded AIS messages are stored the size of the database is about 50% smaller, 12.195 Gigabytes, compared to the 24 Gigabytes.

6.2.2 Query design

The performance of the 8 databases with different data(base) organizations and indexes will be tested by executing designed queries. Queries are designed per defined use case to test whether implementing an organization that is based on use cases is that effective. The queries are designed in *python* while using a connection to the MongoDB database by *pymongo*, the python Application Programming Interface (API) from MongoDB.

Location

The *location* use case asks for the position of a specific vessel at a specified time. The input for a query concerning this use case are the MMSI of the specific vessel and the date-time of which the position is the unknown.
Due to the fact that the update rate of AIS messages are not precisely every second, is not continuous. Therefore it is not possible to query the position of a vessel at an exact time. When asking for an exact time, it might be the case that no message was received and thus no knowledge will exist in the database about the vessels position at that exact time. Querying the position of a vessel at a specific time will be done by asking the last known position of this vessel at the specified time. This position then can be exactly on the time queried or just a few seconds before.
To connect the vessel position with some metadata a second query is specified. This query extracts from the associated metadata message the name and destination of the associated vessel.

The connection between documents of two or more collections in MongoDB can be made by embedding, referencing or by querying twice or more. Both

the embedding of documents as the referencing are explained in section 4.2.3. Both options have the characteristic to multiply the amount of data in one collection by adding data from the other collection to it. To prevent the size of the database from growing, querying the database multiple times to extract all necessary information is chosen as solution to connect the Vessel Position collection with the Metadata collection. Another reason can be found in the fact that with all three queries at first just the MMSI, latitude, longitude and time will suffice, only at a later stadium extra vessel information might be required. Though a combination is tested. Algorithm 6.3 explains the *location* query. It finds the last known location of a vessel at a certain time and combines this location with the by vessel associated metadata.

Algorithm 6.3: Location query

Input: the AIS data from MongoDB
1 **foreach** *position of a vessel before a certain time* **do**
2 Sort the positions date-time descending and limit the output to 1
3 Use the index on **MMSI, date-time** or **no** index to execute this query
4 **for** *the last known position of a vessel at a certain time* **do**
5 Extract the name and destination from the metadata messages
Output: The last known location of a vessel at a certain time linked with vessel name and destination

This algorithm indicates the use of an index. When an index, MMSI or date-time, is used then the by MongoDB provided B-tree index is applied to the MMSI or date-time attributes. More on this subject of indexes is explained in section 6.2.3 and 6.2.3.

Querying the location using Morton codes

Querying the location using the Morton code is done differently. While querying algorithm 5 it is not possible to use the index on the 4D Morton code. Within MongoDB it is only possible to use an index when the attribute of the proposed index is queried. In order to use the Morton code index it is necessary to query the Morton codes. To make this possible, at first the Morton ranges need to be calculated. This means that for the location query the Morton code ranges need to be calculated that are linked to the database records holding the position of a certain vessel before a specified time. These ranges than are to be queried. MongoDB then only has to search these documents holding a Morton code belonging to those Morton ranges to find the last known position. the following algorithm 6.4 will explain how these queries using Morton ranges will be executed.

What this algorithm does not suffices to is how to extract the right Morton code ranges before they can be queried.
[Psomadaki and Martinez, 2016] provides an algorithm that can calculate the Morton codes. This algorithm also provides the reversed calculation, instead of encoding a Morton code, it can decode an existing Morton code into the 2,3 or 4 decimal integers of which it exist. With this ability of encoding and decoding Morton codes, Morton code ranges can be

provided.

Algorithm 6.4: Location query using Morton ranges

Input: Morton codes
1 **foreach** *position of a vessel associated to a calculated Morton code* **do**
2 Sort the positions date-time descending and limit the output to 1
3 Use the index on the Morton code
4 **for** *the last known position of a vessel at a certain time* **do**
5 Extract the name and destination from this vessel
Output: The last known location of a vessel at a certain time linked with vessel name and destination

Even though this algorithm worked well in calculating the Morton code ranges for the *location* query. The fact that the output of the algorithm where the Morton ranges instead of the associated Morton codes, the extraction of the Morton codes from those ranges to make the output implementable in the *location query* was not an optimal fit to the already designed queries.
The focus is on how well the Morton index works, how fast the queries are executed and how many documents are examined (the whole test plan is explained in section 6.2.3). Therefore another more suited option to extract the Morton codes that associate with the Morton ranges is used. The way to 'calculate' the right Morton codes associated to the *location* query (and also the other queries) in this MSc thesis is done by extracting them from the data inside the MongoDB database.
Algorithm 6.5 will explain how the Morton codes to query the *location* query are extracted from the data in MongoDB.

Algorithm 6.5: Finding the Morton codes associated to the *location* query

Input: The Vessel Position messages stored inside MongoDB
1 **foreach** *Morton code within the Vessel Position collection* **do**
2 decode the Morton code
3 **if** *the* MMSI *of the decoded Morton IS EQUAL to the* MMSI *of which the location is wanted AND the date-time of the decoded Morton IS EQUAL OR SMALLER that the date-time of which the location wanted* **then**
4 insert the codes into the Morton *location* query **??**

Trajectory

The *trajectory* use case asks for the trajectory a vessel has sailed. In this MSc research a trajectory of a vessel will be defined as all historic positions of this vessel. Where other studies interpolate the points of a trajectory with each other, here just the single historical positions of the vessel will suffice as historical trajectory. The necessary data entity for a query concerning this use case is the MMSI of the vessel. Algorithm 6.6 explains how querying all positions associated to this MMSI will be done

Algorithm 6.6: Trajectory query

Input: The AIS data from MongoDB
1 **foreach** *position of the specified vessel* **do**
2 Sort the positions date-time ascending
3 Use the index on **MMSI or no index to execute this query**
Output: All the historic positions of the specified vessel

Querying a trajectory using Morton codes

What applies to querying a location with Morton codes applies also to querying a trajectory with Morton ranges. The Morton range codes have to be calculated at forehand and MongoDB only should search the documents containing the right Morton codes. The next algorithm shows how the trajectory of a certain vessel is queried using the Morton ranges.

Algorithm 6.7: Trajectory query using Morton ranges

Input: Morton code
1 **foreach** *position of that specific vessel associated to one of the calculated morton codes* **do**
2 Sort the positions date-time ascending
3 Use the index on the Morton code
4 **foreach** *position within the trajectory* **do**
5 Extract the name and destination of this vessel from the metadata
Output: All historical positions with a associated time stamp linked with vessel name and destination of that specified vessel

The input data for the above algorithm which queries the historical trajectory of a vessel by its associated Morton codes is explained in the following algorithm 6.8.

Algorithm 6.8: Finding the Morton codes associated to the *trajectory* query

Input: The Vessel Position messages stored inside MongoDB
1 **foreach** *Morton code within the Vessel Position collection* **do**
2 decode the Morton code
3 **if** *the MMSI of the decoded Morton IS EQUAL to the MMSI of which the trajectory is wanted* **then**
4 insert the codes into the Morton *trajectory* query **??**

Bounding box

The bounding box use case asks for all unique vessels within a certain geographical area at a specified time. The necessary data to answer this query is the latitude, longitude, date-time and MMSI. For the bounding box query applies the same as when querying the location query, it is not possible to query an exact time. For the bounding box query therefore it is chosen to ask for all vessels located within a geographical area at a time interval of 11 seconds. In this case it is possible to select the vessels that are at anchor which broadcast their position message once every 10 seconds. The follow-

ing algorithm 3 explains how the query will be executed.

Algorithm 6.9: Bounding Box query

Input: AIS data from MongoDB

1 **foreach** *vessel in a specified geoagraphical area at a certain time* **do**
2 give the position of this vessel
3 Use the **x, y, date-time** or **no** index to execute this query

Output: Positions of all vessels inside the sepcified geographical area

Querying all vessels within a bounding box using Morton codes

What applies to querying a location or a trajectory with Morton ranges applies also to querying the vessels within a bounding box with Morton ranges. The Morton range codes have to be calculated at forehand and MongoDB only should search the documents containing the right Morton codes. The 6.10 shows how the vessels within a bounding box are queried using the Morton ranges.

Algorithm 6.10: Bounding Box query using Morton ranges

Input: Morton code ranges

1 **foreach** *position of the vessel inside the specified geographical area at a certain time associated to a calculated morton codes* **do**
2 Extract the location and the MMSI
3 Use the index on the Morton code
4 **foreach** *vessel within the specified geographical area* **do**
5 Extract the name and destination of this vessel from the metadata

Output: All vessels with their positions located within the historical positions at a certain time linked with vessel name and destination of that specified vessel

The input data for the above algorithm which queries the vessels within a bounding box by its associated Morton codes is explained in the following algorithm 4

Algorithm 6.11: Finding the Morton codes associated to the *bounding box* query

Input: The Vessel Position messages stored inside MongoDB

1 **foreach** *Morton code within the Vessel Position collection* **do**
2 decode the Morton code
3 **if** *the latitude and longitude of the decoded Morton IS WITHIN the set bouding box AND the date-time of the decoded Morton IS WITHIN the set timespan* **then**
4 insert the codes into the Morton *bounding box* query 3

6.2.3 Test plan

Testing the performance of the specified and implemented data organization and indexes within MongoDB by comparing the examined amount of documents, the index and database volumes and the response times of the execution of the in section 6.2.2 designed queries.
The performance of the 8 databases with different data(base) organizations and indexes will be tested by executing the three designed queries. A query in combination with a database. For example the execution of the *location* query by the database with a data(base) organization focussed on the use cases holding a day of data is tested on: *response time, the amount of examined documents, and the use of CPU and memory*
Even though the execution of the queries while using the index on the Morton code consists of two parts; extracting the right associated Morton codes and executing the query, testing the effectiveness of the index on the Morton codes does not include the time, amount of examined documents, CPU and memory it takes to extract the Morton code ranges from the database. The effectiveness is measured based on the time, examined documents, CPU and memory it takes to execute the queries when the Morton codes are known.

Different indexes

The performance of a query will at first and foremost be compared with the execution of the same query while using the index on the Morton code. The focus will be on how well this index on the Morton code performs within both of the two different data(base) organizations. The effectiveness of this index therefore is compared to using no index and on using the index on MMSI, latitude, longitude or time.
Where the index on the Morton code was implemented automatically when the AIS data was inserted into MongoDB, the other four indexes on the individual attributes were implemented by the command; **database.collection.ensureIndex(field:ascending/descending)**. This command creates an index on the specified field. All values within this field present in the specified collection and database will then be indexed in the specified ascending or descending order. The comparison of the different indexes includes as explained the response time of the query, the amount of documents that is searched, and the size of the index. Table 6.2 shows these index sizes.

Morton Index	MMSI Index	Date-time Index	X Index	Y Index
328761344	84840448	62828544	112398336	97120331
328257536	84815872	62824448	98630085	93372416
1502334976	459837440	340295680	544497664	509812736
1473912832	459751424	340267008	520912896	499400704

Table 6.2: *Index Volume (KB).*
From top to bottom: Day (spatial-temporal analyses focussed), Day (use case focussed), Week (spatial-temporal analyses), Week (use case focussed)

In order to test the performance of the index on the Morton order, it will be compared to using no index, using an index just on the latitude, using an index just on the longitude, using an index just on the MMSI and using an index just on the date-time. Comparing the performance of the different indexes will conclude on the effectiveness of the Morton code index.

Not all queries are able to use all five indexes. The *location* query can make use of the MMSI index, the date-time and the index on the Morton code. For the *location* query therefore the performance three indexes and no index will be compared. For the *trajectory* query only the performance of the MMSI index, the index on the Morton code and no index are compared. The *bounding box* query can make use of four indexes, the index on the Morton code, the index on the latitude, longitude and date-time. Therefore the *bounding box* query is to be executed five times using a different or no index.
Table 6.3 visualizes the test plan for executing the queries while using different databases and indexes. Such table is used for collecting the response time, the examined documents, the index volume and the CPU and memory. Each of the queries was executed five times to make sure the response time was correct.

		No Index	**Morton**	**MMSI**	**Time**	**X**	**Y**
Location	Day S-T analyses					x	x
	Day use case					x	x
	Week S-T analyses					x	x
	Week use case					x	x
Trajectory	Day S-T analyses				x	x	x
	Day use case				x	x	x
	Week S-T analyses				x	x	x
	Week use case				x	x	x
Bounding Box	Day S-T analyses			x			
	Day use case			x			
	Week S-T analyses			x			
	Week use case			x			

Table 6.3: *Test Plan Queries*

Different Data(base) organizations

The difference in how well the two data(base) organizations perform while executing the three specified queries is first and foremost based on the effectiveness of executing a query while not using any index.

Test and System Environment

All tests, query executions, will be done on one desktop computer which is situated at the nautical test area at Rijkswaterstaat. The circumstances will

be the same for each test run which indicates an equal comparison.
The computer that is used for the tests has a 8.00 GB RAM, an Intel(R) Core(TM) i7 -2600 CPU @ 3.40gHz, and a 64bit operating system.

The parameters for the three queries stay the same when executing the query multiple times.
The location query finds the last known location of the vessel with the name: RWS 16 and MMSI: 244130275 at the first on November 2015 at 13 minutes and 57 seconds past midnight. The trajectory query will find all historical positions of this same vessel and the bounding box query will find all vessels situated around Dordrecht on the first of November 2015 at 01:56.

7 ANALYSIS AND RESULTS

Chapter 6 section 6.2.3 presented a test plan for the comparison of the different data(base) organizations and usable indexes. This chapter will present and discuss the results from the executed performance tests. The first section, section 7.1, will discuss the results associated to the comparison of the used indexes and section 7.2 discusses the results associated to the comparison between the different data(base) organizations. The findings that emerge from these discussions will lead to the conclusion on how to manage AIS data to support spatial-temporal analysis. This will you read in chapter 8

The CPU and the use of memory will not be a part of the performance tests between indexes and data(base) organizations, because both stay with the execution of the three different queries practically equal. MongoDB performs its read operations in-memory, only queries on data that is not in Random Access Memory (RAM) will trigger a read from disk. The use of memory therefore is high, it uses all at the time available memory. The CPU therefore stays low.

MongoDB caches data that is extracted by queries that are often used. When executing a query five times only the first time extracts the data from storage, the other four times will extract the data from cache. In the comparison and analysis of the indexes and data(base) organizations by executing the queries only the response time while executing the queries the first time of the five is used.

7.1 INDEXES COMPARED

The different indexes that can be used to improve the performance will be compared in this section. Most attention will be given to the performance of the index on the Morton code while execution one of the different queries in comparison with other indexes or no index. The performance of the indexes will be tested for each use case; *location*, *trajectory* and *bounding box*.

7.1.1 Location query

The *location* query (designed in section 6.2.2) will find the last known location of a vessel at the first of November 2015 at 13 minutes and 57 seconds past midnight. The query extracts the latitude, longitude, date-time and MMSI from the vessel position collection and the name and destination of the metadata collection. The emphasis is on the extraction of the right position using the vessel positions collection. Figure 7.1 visualizes the outcome of the query in a map.

Figure 7.1: The output of the location query; The location of vessel 244130275 at 01/11/2015 00:13:57

The response time of this query executed while using no index on a day of data with a data(base) organization that would support all possible spatial-temporal analyses is 617,627 seconds. The amount of examined documents is 15.361.707 which is the total amount of documents present in the vessel position collection in the database, a full database scan. The database thus examines one document for the right values according to the *location* query in approximately 40 micro seconds.

Using an index while querying the *location* query has a great influence on the response time and the amount of examined documents. Using the index on the Morton code reduces the response time respectively with 85%-99% depending on the organization of the database (Figure 7.2).

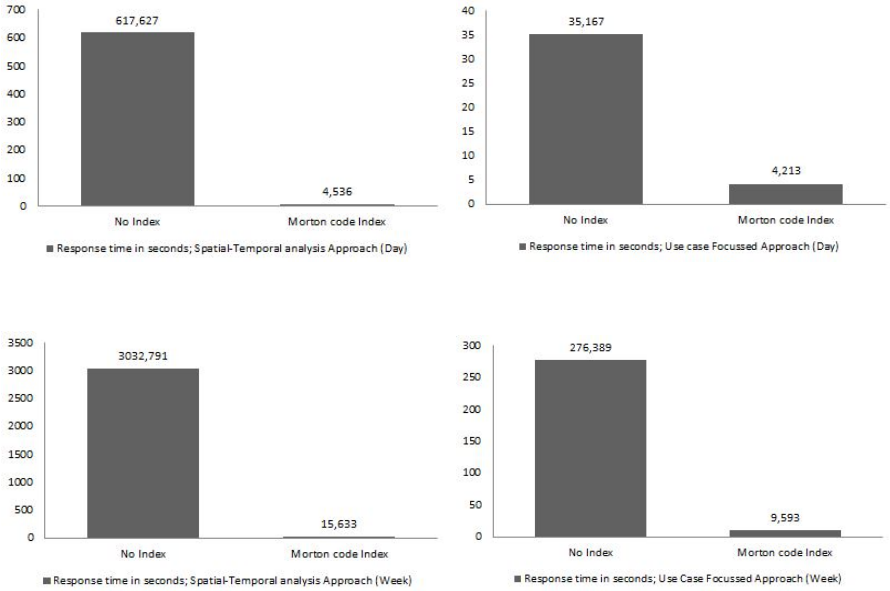

Figure 7.2: Response Time Location Query; No Index vs Morton code Index

The large difference in response time between using no index and using the index on the Morton code could be related to the amount of examined documents. While using the index on the Morton code only 427 documents are examined, using no index means examining all documents within the vessel position collection which is 15.361.707 documents.

The response time while executing the *location* query using the different indexes (Figure 7.3) varies strongly. Using the index on the MMSI improves the response time compared to not using any index (Figure 7.2). Comparing its response time with the ones while executing the *location* query using the date-time and or Morton code index will conclude that using the index on MMSI with the execution of the *location* query is not optimal. A reason for this appearance is found within the amount of documents that is searched with the use of the MMSI index. This is 37.535 documents. This amount is compared to the earlier mentioned 427 documents that are searched using the index on the Morton code a lot. Using the index on the Morton code while querying for the location of a vessel seems besides the least amount of examined documents also to have the fastest response time (Figure 7.3).

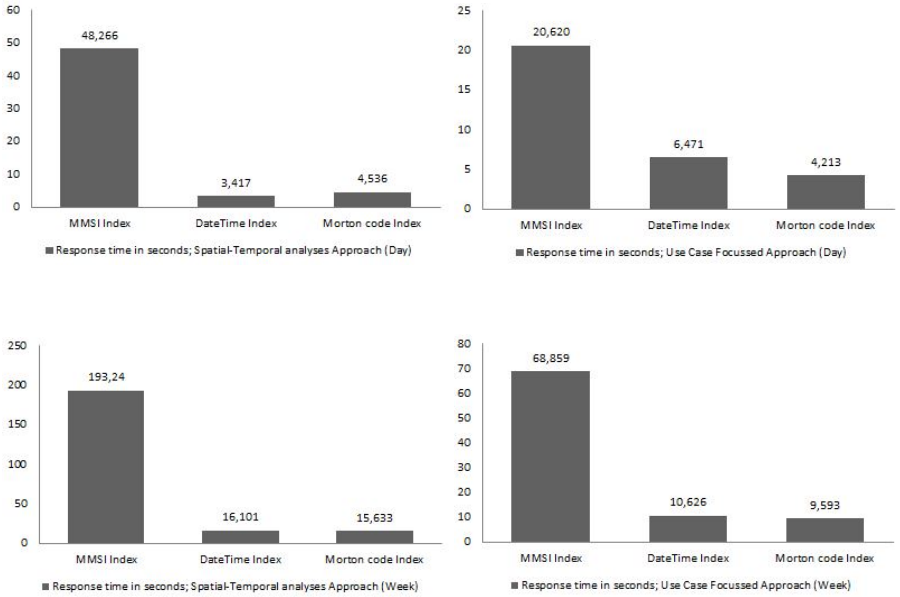

Figure 7.3: Response Time Location Query; MMSI Index vs DateTime Index vs Morton code Index

With the possible existence of a relation between response time and amount of examined documents is mind, when comparing the response time when using MMSI index with the response time of using the date-time index, the amount of documents that are examined with the date-time index should be much lower than the amount examined for the MMSI index use. This assumption, though, is not true. The amount of documents that is examined when the index on the date-time is used is much higher than the amount examined with the MMSI index, namely 328.153 in comparison with 37.535. In this case the assumed relation between response time and amount of examined documents does not add up.
The reason why the response time of the *location* query while using the date-time index is that much faster even though the amount of examined documents is that much higher, lies in the fact that the data is inserted and thus sorted within MongoDB based on date-time. AIS messages that are sent within a few seconds of each other are stored close to each other (assumed clustering, section 5.3). Another point that will substantiate the fast response time of the date-time index is the fact that when MongoDB executes the query when no index is manually proposed, it uses the index that is supposedly the most optimal, which is in this case the date-time index.

The difference in response time between executing the *location* query using the index on the Morton code and using the index on the date-time is only a few seconds (Figure 7.3). The reason for this can also be found in above explanation of the difference between the use of MMSI and date-time index. Only in this case the fact that the amount of examined documents while using the Morton index is only 427 which makes its response time fast, the assumed clustering of the messages by date-time make the use of this index almost equally fast.

Another explanation of the equally fast response time between a *location* query that makes use of the Morton code index and a *location* query that makes use of the date-time index can be found within the influence which the size of the indexes might have on the response time (table 6.2). The index size of the Morton code index is the largest of the three indexes (Morton, MMSI and date-time), the index size of the date-time index is smallest (Figure 6.2). This in combination with the difference between the amounts of examined documents can imply a definitive influence of the index size on the response time.

7.1.2 Trajectory query

This query (designed in section 6.2.2) finds all historical positions, that are available in the database, of one specified vessel. This query extracts the latitude, longitude, date-time and MMSI for each historical position from the vessel positions collection of the database and the name and destination of the vessel are extracted from the metadata collection. Figure 7.4 visualizes the output of the query.

The response time of the *trajectory* query when no index is used is slow, from 30 minutes for a the trajectory of a day to 8 and a half hour to extract the trajectory of one vessel for a week (Figure 7.5). Using no index lead to the examination of all documents available in the vessel positions collection of the database. The larger the amount of documents, the longer it takes to extract all positions of one vessel. Besides the extraction of the right data from the documents, writing the extracted data from the database to a file influence the response time negatively.

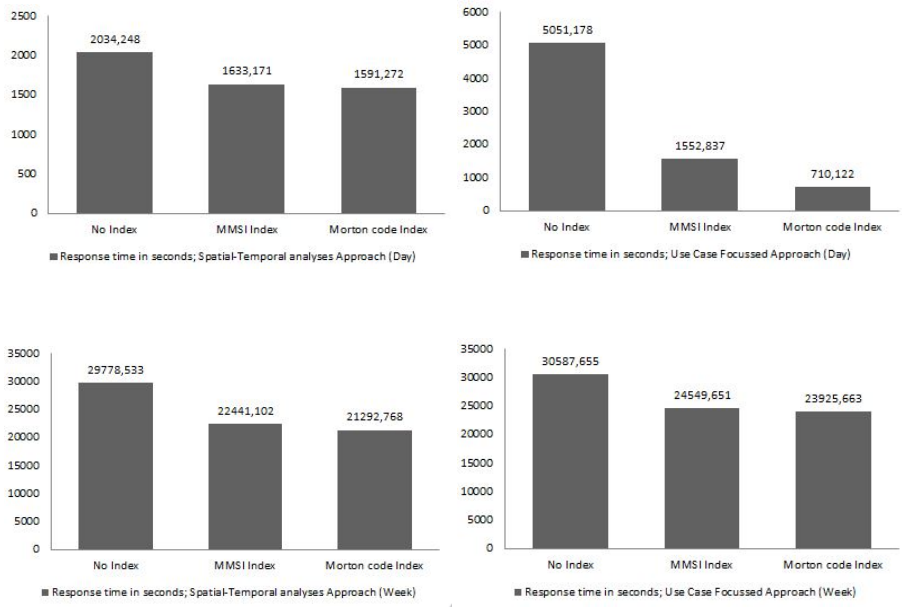

Figure 7.5: Response Time Trajectory Query; No Index vs MMSI Index vs Morton code Index

Figure 7.4: The output of the trajectory query; all historic positions (for a week) of vessel 244130275

Using the index on the Morton code or the MMSI index to execute the *trajectory* query accelerates the response time. According to the drop in the amount of documents that is examined using one of the two indexes compared to the amount of examined documents when no index is used (table 7.1), the response time was expected to be faster.

The difference in response time when using the Morton code index instead of no index is higher for the execution of the *location* query then for the *trajectory* query. The index on the Morton code is thus more effective when multiple attributes of which a Morton code consist are queried together. The trajectory query is focused on extracting the data related to one MMSI where the *location* query is focussed on extracting the data related to one MMSI and a date-time.

Database Volume	No Index	Morton code Index	MMSI Index
Day	15361707 (all documents)	3573	3753
Week	83223293 (all documents)	19688	19688

Table 7.1: *Amount of Examined documents Trajectory query*

Executing the *trajectory* query using the index on the MMSI takes around the same time as when the index on the Morton code was used. The amount of documents that is examined then is the same (table 7.1). The amount of examined documents is the same because only one attribute is asked which means that the search for Morton codes is only influenced by one attribute, the MMSI. All documents containing this specified MMSI are examined both for the use of the Morton code index as for the use of the MMSI. Only when the selection of Morton codes, ranges, is influenced by two or more attributes the amount of documents examined is smaller than when using an index on one of these two or more attributes.

7.1.3 Bounding Box query

This query (designed in 6.2.2) will find all vessels that are situated around Dordrecht at a 01:56 past midnight. The query extracts in this case the MMSI, the latitude and the longitude from the vessel positions collection and the name and destination from the metadata collection. The query output fifty unique vessels in the area of Dordrecht at the specified time (Figure 7.6).

Both the location and the *trajectory* query have a significant faster response time when the Morton code index is used in comparison with the use of no index (Figures 7.2 and 7.5). This applies to the *bounding box* query as well, as long as one day of data is queried. Executing the *bounding box* query using the data volume of one week resulted in response times which conclude that not using any index while executing the query is faster than when the index on de the Morton code is used (Figure 7.7). This conclusion can not be substantiated by the amount of documents that is examined which is with the *location* and *trajectory* use case related to the response time of the query execution. The amount of documents that is examined when the index on the Morton code is used is only 171, where when no index is used all documents within the database are examined.

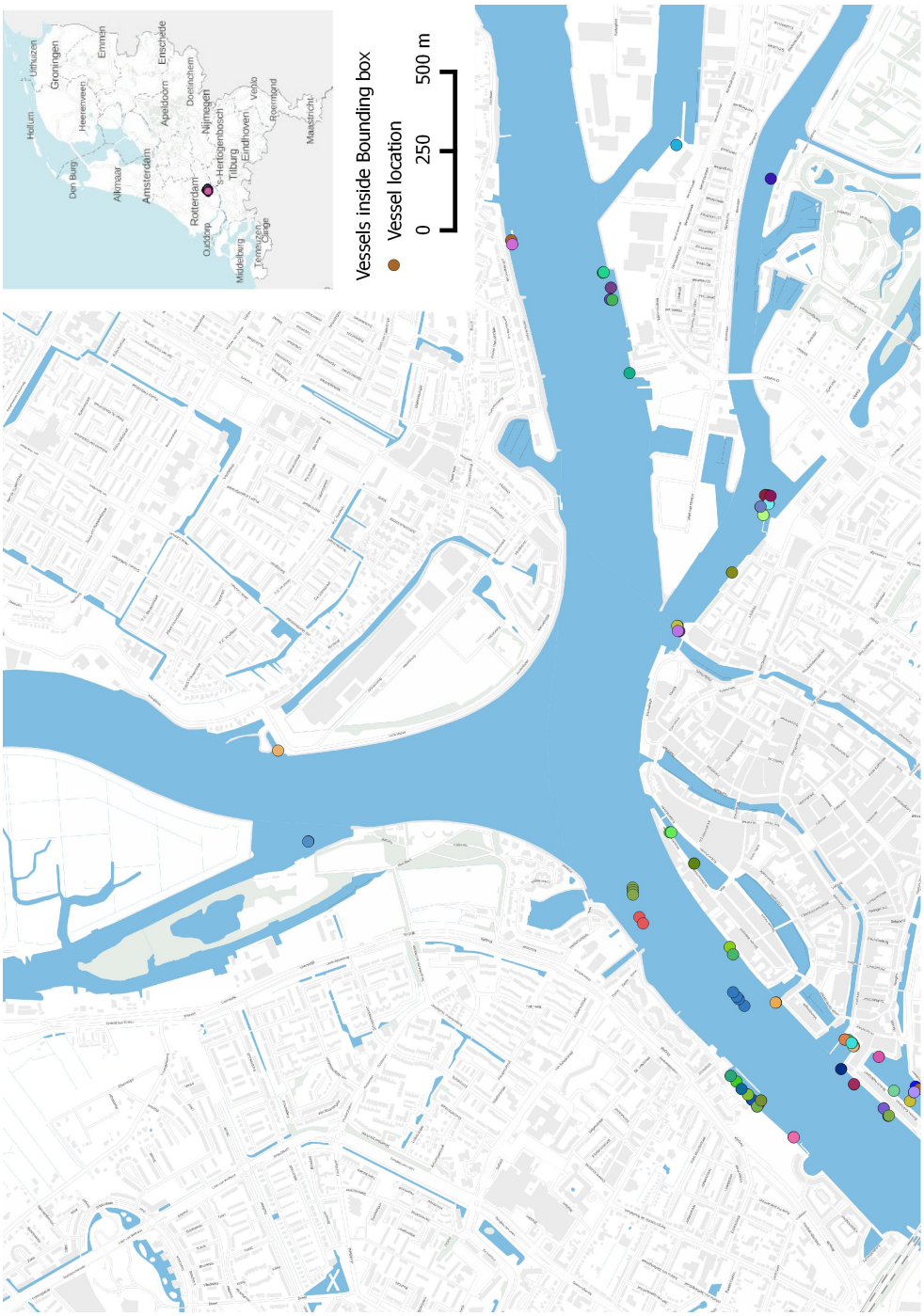

Figure 7.6: The output of the bounding box query; 50 unique vessels around Dordrecht

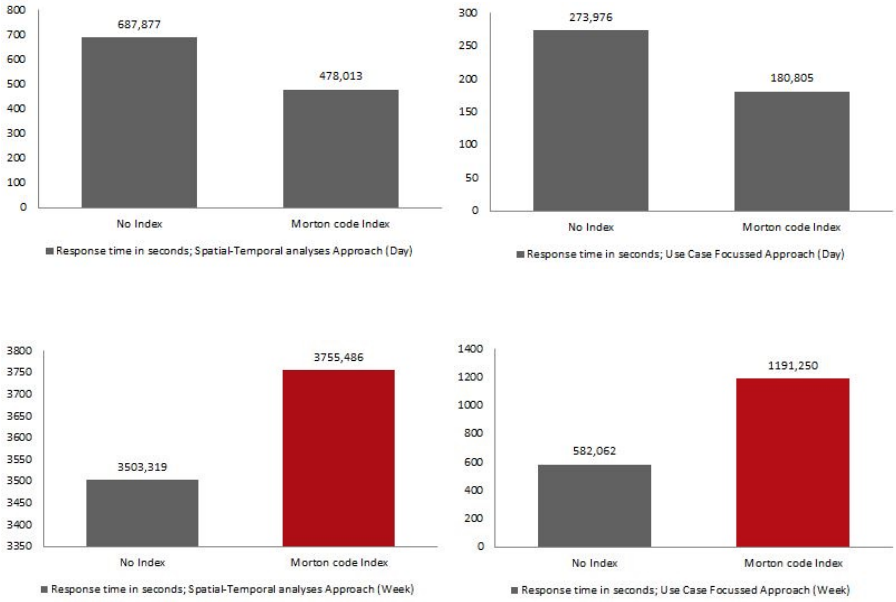

Figure 7.7: Response Time Bounding Box Query; No Index vs Morton code Index

With the execution of the *location* and the *trajectory* query the use of the index on the Morton code made the query most effective in terms of response time and amount of examined documents. For the *bounding box* query using the index on the Morton code does have the lowest amount of examined documents (171) but does not have the fastest response time (Figure 7.8). It can be concluded that the response time for the *bounding box* query using the date-time index is faster than using the Morton code index. This does not apply on the data(base) organization where the focus lies on the use cases and one day of data is implemented though.

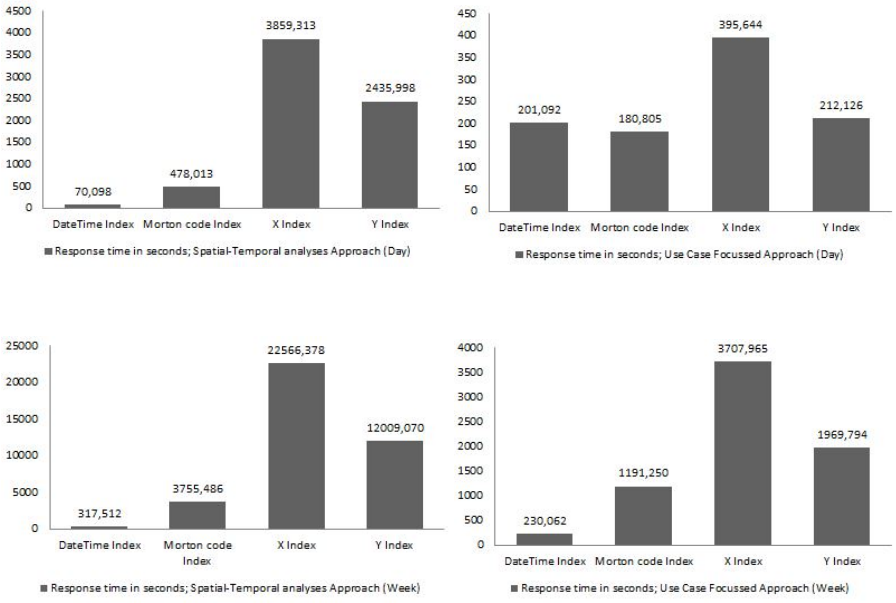

Figure 7.8: Response Time Bounding Box Query; date-time Index vs Morton code Index vs X Index vs Y Index

The reason for the fast response time of the *bounding box* query while using the date-time index is found within the fact that the data is inserted and thus sorted within MongoDB based on date-time. AIS messages that are sent within a few seconds of each other are stored close to each other (assumed clustering, section 5.3). Thi

The differences in response time between the four different indexes while executing the *bounding box* query that exist right now where to be expected. Especially the large difference between the response time when the date-time index is used in comparison with the response time if the index on the X value is used. The date-time interval that is asked within the query is 11 seconds. This 11 seconds is respectively 0.01% of the existing date-time interval within the AIS data of one day.
The interval of X values (longitude) is respectively 3170 meters, which is around 1,25% of the total interval of X values (longitude) within the data (the total width of The Netherlands from East to West).
Because of the difference within the size of the interval according to the data, it could have been expected that the response time of the bounding box when queried using the index on the X value (longitude) was slower than when the date-time index was used.

7.1.4 Concluding remarks compared indexes

Since not one query could be executed by using all five different indexes the concluding remarks are based on the effectiveness of the indexes querying the on of more of the three different queries.
The effectiveness of an index is based on the response time of the query, the amount of documents that is examined, and the volume of the index.
Due to the (assumed) clustering of the data within MongoDB by date-time is the response time of the *location* query using the date-time index fast, only a few seconds slower than when using the index on the Morton code (Figure 7.3), and is the response time of the *bounding box* query fastest while querying it using the date-time index (Figure 7.8). The Size of the date-time index is smallest compared to the other indexes (table 6.2). Even though the data within MongoDB is assumed to be clustered according to the date-time using the index on the Morton code while executing the *location* and *trajectory* query has the fastest response time and lowest amount of documents. Executing the *bounding box* query while using the Morton code index has also a fast response time and the least amount of examined documents.
The use of the Morton code index while executing the three designed queries will therefore be concluded as most effective.

7.2 DATA(BASE) ORGANIZATIONS COMPARED

The response times of the *location* and *bounding box* queries are generally faster for the data organization within the database that is use case focussed than for the data organization focussed on the support for spatial-temporal analyses. This is concluded from the response times of the queries while not using any index. The queries executed on the use case focussed data(base) organization are faster than executing the queries on the data(base) organization focused on broader spatial-temporal analyses (Figure 7.9).

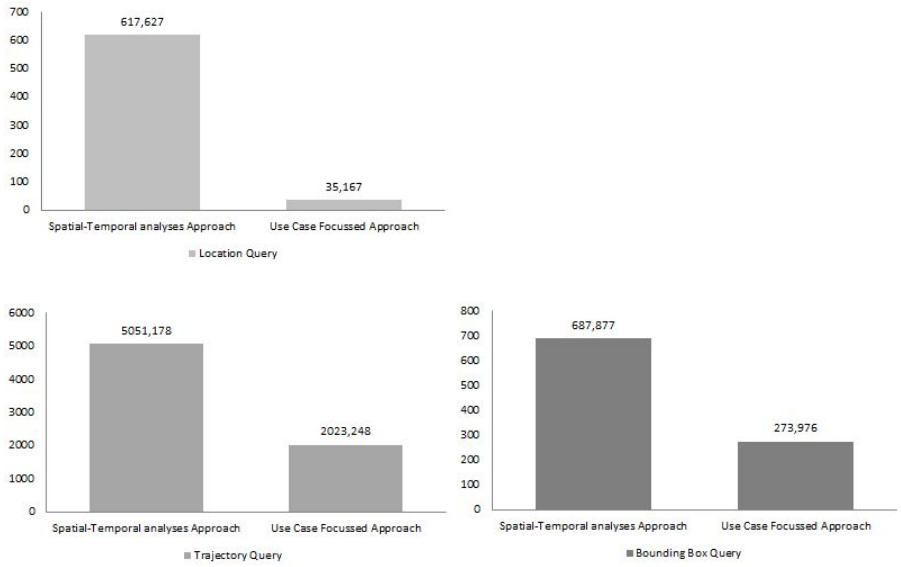

Figure 7.9: Response Times using no Index comparing the two data(base) organizations

The same query on both data(base) organizations examines the same amount of documents in order to find the right key-value pairs.
In order to find the right key-value pairs, which meet the requirements of the query, MongoDB researches each key-value pair within the document in order to find out whether this document contains key-value pairs that meet the requirements. The amount of key-value pairs within a document therefore have influence on the time it takes to find out whether it contains correct pairs.
The vessel positions collection with a data organization based on answering the use cases contain documents which have only six key-value pairs; MMSI, latitude, longitude, date-time, the original message and either the Morton code or an _id given by MongoDB. The vessel positions collection with a data organization for spatial temporal analyses contains documents with 24 key-value pairs; either the Morton code or _id, slot_time-out, sog, repeat_indicator, date-time, slot_number, utc_min, rot_over_range, id, sync_state, rot, true_heading, special_manoeuvre, timestamp, MMSI, raim, spare, utc_spare, nav_status, utc_hour, cog, x (longitude), y (latitude) and position_accuracy. Although this amount depends on the type, id, of the vessel position message.

The difference that exists within the amount of key-value pairs of a document between the two data(base) organizations which is equivalent to the size of both database (section 6.2.1 table 6.1), is proportional to the difference in response time between the two.

7.2.1 Use case focussed data(base) organization and the Morton code index

The date(base) organization focussed on the use cases that where specified can be concluded as most effective in response time. When comparing the effectiveness of the used indexes while using this data(base) organization it can be said that using the index on the Morton code while querying the

location, trajectory or *bounding box* query is most effective in its response time and amount of examined documents.

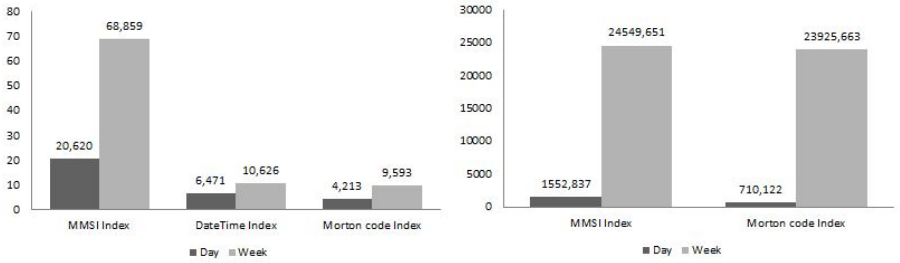

Figure 7.10: Response Times Location and Trajectory query Use Case focussed data(base) organization using the index on the Morton code

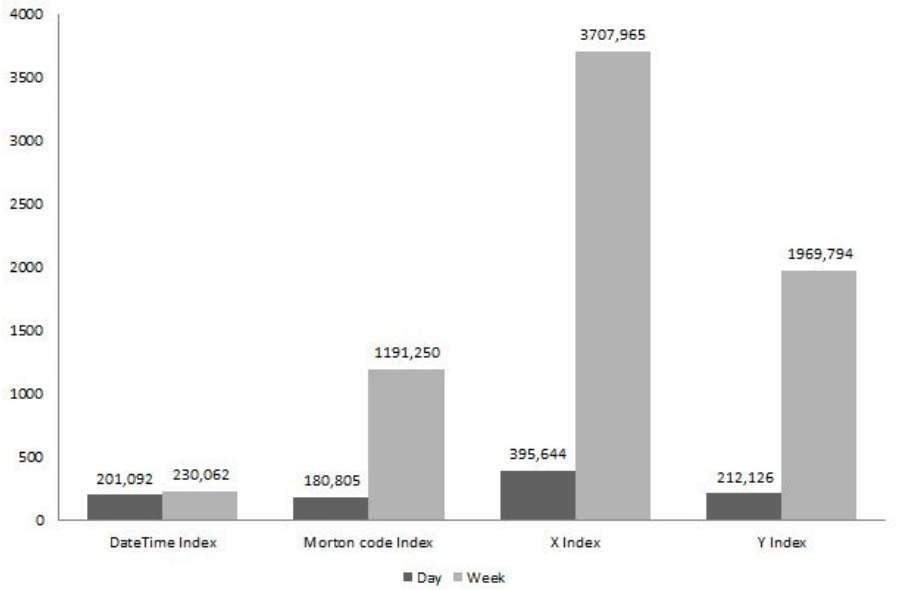

Figure 7.11: Response Times Bounding Box query Use Case focussed data(base) organization using the index on the Morton code

Concluding remarks

The data(base) organization based on the Use Cases is fast in the extraction of the information of where the organization is designed for. Extraction of other AIS information is possible but only by decoding the original NMEA encoded AIS message.
When the use cases on which the data(base) organization is based are the only ones or other use cases are also only based on the extraction of the four pieces of information (latitude, longitude, MMSI and date-time) then organizing the database according to Use Case Focussed Approach is most

effective. But effectively querying other information for other use cases might then desire another data(base) organization, which means that the AIS data will be stored twice or more times.
AIS data within MongoDB according to the data(base) organization based on the Spatial-Temporal analyses Approach can easily be used for all available use cases, no other storage organizations are necessary when new use cases are presented.

8 CONCLUSION

This thesis aimed to provide a management, storage, and structuring solution for historic AIS data to support spatial-temporal analyses.
Drawing conclusions on a suitable management solution will be done by first answering the sub research questions followed by the main research question of this MSc thesis. The conclusion will finish with a discussion on the methodology and recommendations for future work.

8.1 RESEARCH QUESTIONS

Five sub questions and a main research question are posed in this research (section 1.3).

What is AIS data, what are its features?
AIS data can be referred to as a large and growing volume of semi-structured data containing out of digital NMEA-0183 encrypted messages vessels sent towards other vessels and base stations and base stations send to vessels. Two radio channels especially reserved for AIS are used to sent these messages. AIS messages can be defined as real-time data, containing real-time positions of vessels, it is therefore mainly used by VTS.

A vessel can sent 27 different kind of messages of which the update rate and schema is different. Each message a vessel sent is related to each other by the MMSI number of the associated vessel. There exist six types of messages; position messages, metadata messages, addressed messages, broadcast messages, acknowledgement messages, and defined slot binary messages. Between the different kind of AIS messages there exist a difference in update rate. The update rate of the vessel position messages for example depends on the speed of the vessel, when it reaches a high speed the vessel position messages will automatically be sent once every two seconds. For the metadata messages applies another update rate, once every three to six minutes.

What kind of spatial-temporal analyses with historic AIS data is interesting (for Rijkswaterstaat)?
Rijkswaterstaat has formulated use cases of which three give an indication of the bigger picture of spatial-temporal analyses. The first proposed use case is one concerning the historic location of a vessel. The most important question among the spatial-temporal analyses is where was a vessel positioned at a specific time in history. This question, use case, is the base for further spatial-temporal analyses. The second use case is an elaboration of the location use case, namely the request for all historical locations of a specific vessel. This use case is based on the idea of a historical trajectory. Such as where is a vessel been over the last

few days, weeks, months. The third use case is also an elaboration on the location use case. This use case is about finding all vessels that are positioned at a certain time in a specified bounding box, a geographical area.

Even though these three use cases give an indication on what Rijkswaterstaat would like to do with the historical AIS data after it is properly managed, there exist many more possible use cases. A broader spectra of spatial-temporal analyses that are able with the historical AIS data should not be excluded.

What database should store historic AIS data?
A database suitable for the management of AIS should be able to store a large volume of semi-structured data that is kept up-to-date by frequent (real-time) updates. Where the traditional relational databases are pushed to their limits with respect to handling the volume and update rates of the AIS data, are NoSQL databases designed to handle large volumes of data by their ability of horizontal scaling and is for these databases the update rate of the data less of a problem.
There are four kinds of NoSQL databases of which two are able to manage AIS data, column family and document store databases. From these two kinds of NoSQL databases seemed MongoDB based on literature and database documentation as the database that should store AIS data. MongoDB is a document store database which easily stores large volumes of data without a fixed scheme. MongoDB is a database which made use of the features of traditional relational databases it had optimized such as the ability to create and execute complex queries and the ability to create multiple secondary indexes. MongoDBs flexibility (with scheme) horizontal scalability, fast read and multiple write performance and its ability to execute complex queries make MongoDB suitable to store AIS data

How should this database (MongoDB) store the historic AIS data?
The organization of the data(base) depends primary on the use cases and the broad spectra of spatial-temporal analysis which is possible to perform on the historical AIS data, which has led to the implementation of two data(base) organizations. One especially focussed on the use cases whereby the addition of the original AIS message the reason is that it is possible to perform spatial-temporal analyses, and one data(base) organization focussed on the support for all spatial-temporal analyses including the use cases. Secondly how this organizations are implemented, is related to the use cases in combination with the data features.

Storing the AIS data in MongoDB is done by the insertion of JSON files containing either vessel position messages, metadata messages or all other kinds of messages into three different collections. This division of AIS messages is made based on the by the use cases necessary and requested data. This data is found within the vessel position messages or in the metadata messages. Because data is not supposed to be wasted a third collection is made with all other messages.
The data(base) organization focussed on the use cases stores within the three collections only the MMSI, latitude, longitude, date-time and the original AIS messages. The data(base) organization focussed on spatial-temporal analysis stored within the three collections all available decoded AIS data.

A comparison between the two data(base) organizations is done by executing the three designed queries (to support the use cases) and comparing the response time of these queries and amount of examined documents. The data(base) organization focussed on the use cases can be concluded the most efficient data(base) organization of the two. The small size of the database is to be the reason for this outcome.
Storing AIS data in MongoDB should be done according to the data(base) organization focussed on the three specified use cases.

What indexing technique is suitable to provide efficient historic spatial-temporal data requests?
The 4D Morton code SFC is used as a 'proof of concept' and is found quite effective in the indexing of vessel positions. Even though no complete Morton SFC based clustering of the AIS was introduced in MongodB (clustering was based on date-time), the B-tree index on the 4D Morton code enables a fast look up of the latitude, longitude, MMSI and the date-time of the vessel positions.
Other B-tree indexes were proposed on the four attributes (latitude, longitude, MMSI and date-time) individually. While comparing the different indexes it can be concluded that with the use of the index on the 4D Morton code all three queries were executed with a fast response time and just a few documents had to be examined in order to find the right information.
The efficiency of using the index on the date-time though came close to the efficiency of the 4D Morton code index due to the small intervals of just a few seconds that were queried and the clustering of the data based on the date-time values due to the insertion of the data into MongoDB.
Using the index on the Morton code is concluded most effective, but what must be kept in mind is the fact that before a query can be executed with the use of the Morton index several steps are to be taken. The extraction or calculation of the right Morton order codes that are associated to the query and the design of the query extracting the right information consisting these Morton codes to be able to use the index based on these codes. To increase the effectiveness of the Morton code index a clustering of the data according to this code is desired.

After having answered the five sub questions the main question can be addressed. ***How can historic AIS data be managed, stored, and structured to support spatial-temporal data analyses?***
The organization of the data(base) with the focus on answering the by Rijkswaterstaat specified use cases in combination with an index on the data based on the calculated 4D (latitude, longitude, MMSI and date-time) Morton code inside MongoDB is the AIS data management solution this MSc thesis proposes.

MongoDB allows the AIS messages to be individually stored in documents which together will form a collection (of AIS messages). The data(base) organization structures the AIS data into three collections each containing either MMSI, latitude, longitude, date-time and original AIS message concerning vessel positions or the original AIS message, the MMSI, date-time destination, name, draught, type and cargo concerning metadata messages or the MMSI, date-time and original AIS message of all other remaining AIS messages. Organizing the data like this in three collections only containing a few attributes while not throwing away data, keeps the

size of the database small.
The execution of the queries that are designed to support the specified use cases therefore is fast and searches a small amount of documents for the right information when using the B-tree index on the 4D Morton code.

8.2 DISCUSSION

This discussion is used to review the methods that are used in this MSc research. The overall used method based on Klein et al. [2015] shall not be questioned, only the methods used within the by Klein et al. [2015] defined stages.
MongoDB was chosen based on literature and database documentation. A database which should be te right choice on paper does not have to be the right choice in reality. In this case the choice and use of MongoDB worked out well. Though, mentioned within the related work (chapter 2), performing a benchmark research where different databases are tested and compared in practise will substantiate the choice for database based on practice. The choice for a database based on the outcome of a use case specific benchmark research is well substantiated. From a choice for a database based on literature and database documentation is it not clear before usage whether it is in practise the right choice as well.
The scope of this MSc research indicates that the real-time update rate of AIS data must be taken into account in the choice for database and indexing technique. MongoDB presents itself as a database in which updates do not have to be that difficult, therefore the ability to handle real-time AIS data is assumed to be suitable. As is the same with the index on the Morton code. Te use of the index on the Morton code is not adapted towards the ability to use it in a historical AIS database with a real-time update. For this, space on the hyper cube should be reserved for all AIS messages that will go eventually into the database Psomadaki et al. [2016].
The AIS data that is used consists of one day and one week of data which is not comparable with the volume of data that the historical database eventually will hold. Though this volume is seemed efficient enough to draw conclusions about the data(base) management, storage and structuring of AIS data.

8.3 FUTURE WORK

There are several directions for future work which can be derived from this MSc research. These can be defined as direct extensions of this research. This section will briefly explain these proposed directions.

Using a Hilbert SFC
The 4D Morton SFC was used as a 'proof of concept' which has proven to be an efficient method for the indexing of AIS data. Lawder and King [2001]; Moon et al. [2001] suggest the Hilbert SFC to be more efficient than the Morton SFC. An improvement to the used method for managing AIS data therefore could be the use of the Hilbert SFC for cluster/index of the data.

Using an enhanced version of the B-tree
Section 2.3.1 discusses the by Chen et al. [2008b] proposed Bx-tree which is seen as the first attempt to adapt the B-tree to the indexing of moving objects. MongoDB uses the standard implemented B-tree which in this MSc research is used on a static dataset. An improvement to the used index on a static dataset would be to use the Bx-tree with a dataset which is updated in real-time.

Using another database
The discussion (section 8.2) already implies the necessity for a benchmark research for the purpose of finding the most suitable database to use for the management of AIS data. To make certain the choice for MongoDB that was made is respectful, the performance of another database while storing and structuring AIS data should be tested. A comparison between the performance of for example the databases CASSANDRA or PostgreSQL while managing, storing and structuring AIS data and MongoDB could be an interesting research area.

Clustering with the Morton SFC
The clustering of the data is assumed to have been done on the date-time. To ensure a more effective data(base) organization the clustering should accomplish the indexing technique. Which in this case was the 4D Morton SFC. Ensuring the effectiveness of a cluster will be an interesting next research.

BIBLIOGRAPHY

Abramova, V., Bernardino, J., and Furtado, P. (2014). Experimental evaluation of nosql databases. *International Journal of Database Management Systems*, 6(3):1.

AISM and IALA (2005). Iala guideline no. 1050 on the management and monitoring of ais information. *Guideline 1050 on the Management and Monitoring of AIS information, Edition 1*.

Amirian, P., Winstanley, A., and Basiri, A. (2013). Nosql storage and management of geospatial data with emphasis on serving geospatial data using standard geospatial web services.

Baas, B. (2012). Nosql spatial: Neo4j versus postgis. Master's thesis, GIMA, TU Delft, Delft University of Technology.

Brakatsoulas, S., Pfoser, D., and Tryfona, N. (2004). Modeling, storing and mining moving object databases. In *Database Engineering and Applications Symposium, 2004. IDEAS'04. Proceedings. International*, pages 68–77. IEEE.

Campbell, P. M., Devine, K. D., Flaherty, J. E., Gervasio, L. G., and Teresco, J. D. (2003). Dynamic octree load balancing using space-filling curves. *Williams College Department of Computer Science, Tech. Rep. CS-03-01*.

Chen, S., Jensen, C. S., and Lin, D. (2008a). A benchmark for evaluating moving object indexes. *Proceedings of the VLDB Endowment*, 1(2):1574–1585.

Chen, S., Ooi, B. C., Tan, K.-l., and Nascimento, M. A. (2008b). St 2 b-tree: a self-tunable spatio-temporal b+-tree index for moving objects. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 29–42. ACM.

Chon, H. D., Agrawal, D., and Abbadi, A. E. (2001). Using space-time grid for efficient management of moving objects. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 59–65. ACM.

Diallo, O., Rodrigues, J. J., Sene, M., and Lloret, J. (2015). Distributed database management techniques for wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 26(2):604–620.

Duntgen, C., Behr, T., and Guting, R. H. (2009). Berlinmod: a benchmark for moving object databases. *the VLDP Journal*, 18(6):1335–1368.

Frentzos, E. (2008). Trajectory data management in moving object databases. *Phd Thesis, University of Piraeus*.

Guting, R. H. and Schneider, M. (2005). *Moving Objects Databases*. Morgan Kaufmann, Elsevier.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE.

Hecht, R. and Jablonski, S. (2011). Nosql evaluation: A use case oriented survey. In *International conference on cloud and service computing*, pages 336–41. IEEE.

Hussien, E. O. E. (2012). Querying moving objects database. *CU Theses, Cairo University*.

Indrawan-Santiago, M. (2012). Database research: Are we at a crossroad? reflection on nosql. In *2012 15th International Conference on Network-Based Information Systems*, pages 45–51. IEEE.

Jensen, C. S., Lin, D., and Ooi, B. C. (2004). Query and update efficient b+-tree based indexing of moving objects. In *Proceedings of the thirtieth international conference on very large data bases*, volume 30, pages 768–779. VLDB Endowment.

Kashyap, S., Zamwar, S., Bhavsar, T., and Singh, S. (2013). Benchmarking and analysis of nosql technologies. *Int J Emerg Technol Adv Eng*, 3:422–426.

Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., and Matser, C. (2015). Performance evaluation of nosql databases: A case study. In *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, pages 5–10. ACM.

Klein, N., Mossop, J., and Rothwell, D. R. (2009). *Maritime security: international law and policy perspectives from Australia and New Zealand*. Routledge.

Lawder, J. K. and King, P. J. H. (2001). Querying multi-dimensional data indexed using the hilbert space-filling curve. *ACM Sigmod Record*, 30(1):19–24.

Lema, J. A. C., Forlizzi, L., Guting, R. H., Nardelli, E., and Schneider, M. (2003). Algorithms for moving object databases. *The Computer Journal*, 46(6):680–712.

Li, Y. and Manoharan, S. (2013). A performance comparison of sql and nosql databases. In *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pages 15–19. IEEE.

Lin, D. (2007). *Indexing and querying moving objects databases*. PhD thesis, National University of Singapore.

Lourencco, J. R., Cabral, B., Carreiro, P., Vieira, M., and Bernardino, J. (2015). Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of Big Data*, 2(1):1–26.

Meng, X. and Chen, J. (2011). *Moving Objects Management: Models, Techniques and Applications*. springer Science & Business Media.

Mokbel, M. F., Aref, W. G., and Kamel, I. (2003a). Analysis of multi-dimensional space filling curves. *Geoinformatica*, 7(3):179–209.

Mokbel, M. F., Ghanem, T. M., and Aref, W. G. (2003b). Spatio-temporal access methods. *IEEE Data Eng. Bull.*, 26(2):40–49.

MongoDB (2016). Mongodb architecture guide. Technical report.

Moniruzzaman, A. and Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison.

Moon, B., Jagadish, H., and Faloutsos, C. (2001). Analysis of the clustering properties of the hilbert space filling curve. *Transactions on knowledge and data engineering*, 13(1):1–18.

Naisbitt, J. and Cracknell, J. (1982). Megatrends: Ten new directions transforming our lives. Technical report, Warner Books New York.

Nguyen-Dinh, L.-V., Aref, W. G., and Mokbel, M. (2010). Spatio-temporal access methods: Part 2 (2003-2010).

Nittel, S. (2015). Real-time sensor data streams. *SIGSPATIAL Special*, 7(2):22–28.

Park, Y., Liu, L., and Yoo, J. (2013). A fast and compact indexing technique for moving objects. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference*, pages 562–569. IEEE.

Parker, Z., Poe, S., and Vrbsky, S. V. (2013). Comparing nosql mongodb to an sqlogical db. In *Proceedings of the 51st ACM Southeast Conference*, page 5. ACM.

Pfoser, D., Jensen, C. S., Theodoridis, Y., et al. (2000). Novel approaches to the indexing of moving object trajectories. In *Proceedings of VLDB*, pages 395–406.

Plugge, E., Hows, D., Membrey, P., and Hawkins, T. (2015). *The definitive guide to MongoDB: A complete guide dealing with Big Data using MongoDB*. Apress.

Politie (2016). Vaarregels. https://www.politie.nl/themas/vaarregels.html. Accessed 2016-09-18.

Psomadaki, S. and Martinez, O. R. (2016). Morton. https://github.com/stpsomad/DynamicPCDMS/blob/master/pointcloud/morton.py. Accessed 2016-09-18.

Psomadaki, S., Van Oosterom, P., Tijssen, T., and Baart, F. (2016). Using a space filling curve approach for the management of dynamic point clouds.

Reiss, F., Stockinger, K., Wu, K., Shoshani, A., and Hellerstein, J. M. (2007). Enabling real-time querying of live and historical stream data. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 28–28. IEEE.

Ristic, B., Scala, B. L., Morelande, M., and Gordon, N. (2008). Statistical analysis of motion patterns in ais data: Anomaly detection and motion prediction. In *information fusion, 2008 11th international conference on*, pages 1–7. IEEE.

Rodríguez-Tastets, M. A. (2005). Moving objects databases.

Santos, P. O., Moro, M. M., and Davis Jr, C. A. (2015). Comparative performance evaluation of relational and nosql databases for spatial and mobile applications. In *Database and Expert Systems Applications*, pages 186–200. Springer.

Schwehr, K. (2015). libais. https://github.com/schwehr/libais.com.

Sistla, A. P., Wolfson, O., Chamberlain, S., Dao, S., et al. (1997). Modeling and querying moving objects.

Sullivan, D. (2015). *NoSQL for mere mortals*. Addison-Wesley.

Tetreault, C. B. J. (2005). Use of the automatic identification system for maritime domain awareness (mda). In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 1590–1594. IEEE.

Van der Veen, J. S., Van der Waaij, B., and Meijer, R. J. (2012). Sensor data storage performance: Sql or nosql, physical or virtual. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 431–438. IEEE.

van Oosterom, P. (1999). Spatial access methods. *Geographical information systems*, 1:385–400.

Wang, L., Zheng, Y., Xie, X., and Ma, W.-Y. (2008). A flexble spatio-temporal indexing scheme for large-scale gps track retrieval. In *The Ninth International Conference on Mobile Data Management (mdm 2008)*, pages 1–8. IEEE.

Wijaya, W. M. and Nakamura, Y. (2013). Predicting ship behavior navigating through heavily trafficked fairways by analyzing ais data on apache hbase. In *computing and networking (CANDAR), 2013 first international symposium on*, pages 220–226. IEEE.

Wolfson, O., Xu, B., Chamberlain, S., and Jiang, L. (1998). Moving objects databases: Issues and solutions. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 111–122. IEEE.

Xia, Y. and Prabhakar, S. (2003). Q+ rtree: Efficient indexing for moving object databases. In *Database Systems for Advanced Applications, 2003. (DASFAA 2003). Proceedings. Eight International Conference on*, pages 175–182. IEEE.