Universität Rostock

Traditio et Innovatio

Bachelor Thesis

Transformation of Matlab analysis programs into database-supported evaluations of SQL and NoSQL database systems

eingereicht von: Jakob Voigt

eingereicht am: 11.02.2020

Gutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
Dr.-Ing. Holger Meyer

Betreuer: Dr. Ralf Prien
Mareike Floth-Peterson
M.Sc. Tanja Auge

Abstract

In the recent years, data has dramatically increased not only in amount but also in value. Therefore, a growing number of scientific institutions counts on database systems benefiting from less memory usage, prompt real-time analysis and protected security. This paper examined how Matlab-analysis can be potentially transformed into database-supported evaluations in SQL and NoSQL systems. Thereby, this work focuses on the Matlab analysis that is performed by researchers of the Leibniz-Institute for Baltic Sea Research working with highly sensitive physical sensor-data gathered by a sampling station in the Godland Basin. Due to the highly sophisticated analysis, a simplified reference model was developed to illustrate the transformation. Furthermore, an exemplary data model was constructed to represent the broad variety of semi-structured data. Based on the gathered data and the Matlab scripts, two examples were included in the reference model. This involved sorting and polynomial evaluation for computing conductivity. Based on the latest state of art and the data properties, three individual databases were chosen for the transformation. As a relational DBMS, PostgreSQL has been picked. Furthermore, two non-relational databases were chosen: Apache's Cassandra as column store and MongoDB as document store. Generally, it could be shown that no database is capable to transform the entirety of performed Matlab functionalities on its own, but relational databases seem to be the best fit. Therefore, it is recommended to launch a holistic approach that combines the benefits of both Matlab and relational databases.

Contents

List of Acronyms

BenthosDB	Benthos Database
BSON	Binary JSON
CQL	Cassandra Query Language
DBaaS	Database as a Service
GODESS	Gotland Deep Environmental Sampling Station
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IOW	Leibniz Institute for Baltic Sea Research Warnemünde
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
PIP	Profiling Instrument Platform
RDBMS	Relational Database Management Systems
SQL	Structured Query Language
UDF	User-Defined Function
UnQL	Unstructured Query Language

List of Figures

List of Tables

Listings

1. Introduction

Big data is on the rise. But not only that data is growing in concerns of quantity, it is also enormously increasing in value. Simultaneously, the quality of big amounts of data is decreasing. Thus, finding appropriate methods for processing these highly precious amounts of unstructured data has become a greatly significant issue. Thereby, the data is generated by multiple sources such as sensor systems. Since they are getting cheaper, smaller and easier to maintain, the deployment of sensors is increasing and they are used in a growing number of applications in both industry and the scientific sector. [VVM12] However, not every company that is making use of sensor technology stores the gathered data in a database-supported way. Thus, data processing can be highly storage as well as time-consuming. As a consequence, companies and scientists are searching for solutions for storing their data in a more efficient way. While some prefer to store the data in traditional SQL-based databases due to their benefits in consistency, others prioritize NoSQL databases because of their high flexibility and information promptness. [SK11]

One German research institute that finds itself strongly involved in the processes of this development is the Leibniz Institute for Baltic Sea Research Warnemünde (IOW). In the course of several projects, the institute collects biological, chemical, physical and geological data about different environmental parameters of the Baltic Sea. The data is gathered either by fixed profiling stations or by maritime field trips. Due to the high variety of heterogeneous data, the IOW sees potential for improvement in the way the data is stored and processed. In cooperation with the Database Research Group (DBIS) of the University of Rostock, the IOW is looking for concepts to face this problem. [BKM+17] Therefore, the DBIS group supports the IOW in order to homogenize this eclectic shaped diversity of information in the long-term. The data is generated by a bunch of third-party funded projects. Thereby, the Benthos Database (BenthosDB) guided by the research group "Ecological benthic organisms" and the Gotland Deep Environmental Sampling Station (GODESS) supervised by the research group "Chemical in-situ sensors" belong to the most important undertakings at the IOW. While BenthosDB already uses MySQL as database-supported platform, the analyses performed by the GODESS project still remain unsupported. Therefore, this paper will focus on the analyses conducted on the GODESS data. For processing this data, researchers are using Matlab which is a widely used mathematical analytics and visualization software.

The aim of this thesis is to analyze how the Matlab analysis programs used within the scope of the GODESS project and implemented by scientists of the IOW can potentially be transferred into database-supported evaluations on both SQL and NoSQL database systems. Thereby, the purpose is to transform fractions of the Matlab analysis into various database types in order to demonstrate the benefits and drawbacks of each type. In chapter 2, it will be analyzed, how the gathered data is structured and processed in Matlab. In chapter 3, this work will furthermore provide an overview about the latest state of art involving current database systems and discuss the advantages and disadvantages as well as application areas for SQL and NoSQL platforms. In chapter 4, it will be further discussed which parts of the Matlab analysis are considered to be realizable on a database level and which not. In chapter 5, some identified functions will be finally implemented in an exemplary manner on different database systems including both SQL and NoSQL platforms. In doing so, PostgeSQL is used as a SQL-based platform whereas MongoDB and Cassandra are used as representatives of NoSQL databases.

2. Background

Like many scientific and economic institutions the IOW has gathered a broad variety of heterogenic data over the recent hundred years. This data was gathered by different projects, profiling stations and research ships. Due to the high heterogeneity of data, there is an increasing need of homogenization as well as new storing techniques. Therefore, the IOW cooperates with the University of Rostock. The purpose is to create a pipeline for integrating the heterogeneous measurement data into database-supported solutions that is taking account of both data provenance and temporal aspects.[BKM+17] The IOW has adapted MySQL as a representative for relational database systems for some of their projects like, for instance, the BenthosDB. This project gathers data on a research vessel navigated on several routes across the Baltic Sea. In contrast to the Benthos project, the GODESS is embedded in the Gotland Basin and collects the data within a fixed spot. In case of the GODESS profiling station, there is no database-supported solution, yet. The analyses are still performed on a file level which means that files are stored locally and not centralized on a database server.

Consequently, every user who wants to work on those files needs to copy the files to his terminal which creates redundancy as well as storage issues. Since there is no historization of changes, the formerly exerted methods lack in terms of data provenance and in the backtracking of file changes. An adequate data provenance would imply that every step of the analyses can be potentially traced without extensive effort. Since these requirements are not fulfilled at the moment, there are difficulties to reconstruct changes made by the users. In addition, privacy aspects are realized only on a low granular level. These aspects include both the privacy of the user's personal data as well as the security of the data used for the analysis.

The smallest possible granularity that can be realized is the file level. Furthermore, concurrent multi-user operations are currently impossible. Additionally, it is impossible to allow users to read or view individual values from the files without firstly editing the files. Thus, access to the entire data file has to be enabled or permitted. All in all, these arguments underline the need for database-supported solutions which are considered to be less storage and time-consuming when it comes to semi- or unstructured data. Performing the Matlab analysis or at least some parts of it, on a database level could potentially lead to a higher information promptness and decrease main memory usage. Before we can discuss which types of databases will be suitable in this application, we firstly need to understand how the sampling station works, how the measured data is structured and how the performed Matlab anaylsis programs are scripted.

2.1. The GODESS Project

As a profiling embedment located in the Godland Basins the GODESS constitutes a measuring platform for investigating environmental parameters. In particular, physical and chemical variables in different depths of the Baltic Sea are measured. The sensor system of the sampling station is situated on a so called Profiling Instrument Platform (PIP). Thereby, the data is generated in-situ which means that information is obtained directly on site and not influenced by external parameters. The following section

will explain the functional design as well as the basic instruments used for measuring the parameters as an entry point for understanding how the data is processed. Figure 1 illustrates how the GODESS is constructed and embedded in the Godland Basin. Thereby, it shows of which components the GODESS consists and how they act in concert. Basically the components split into the PIP (**A**), an underwater winch (**B**), a bottom weight (**C**) and an acoustic releaser (**D**).

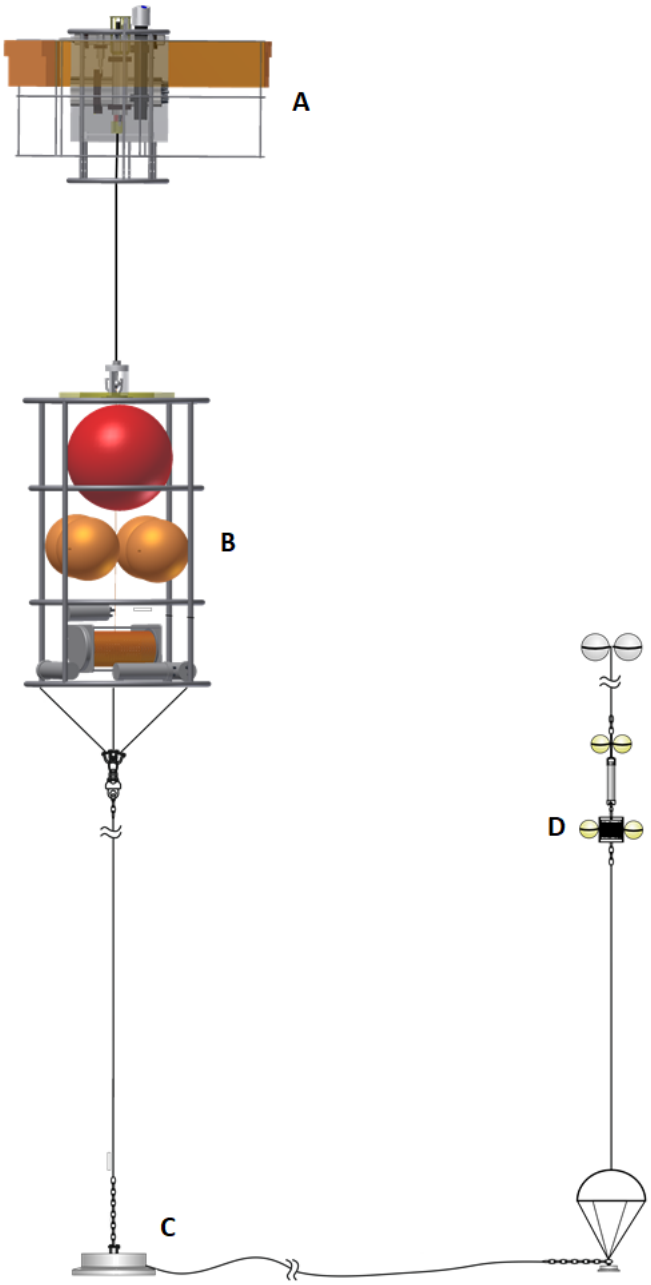

Figure 1.: Graphical Illustration of the GODESS [Pri19]

The GODESS operates in the depth interval from 180 meters till 30 meters below the water surface. In a depth of 220 meter below the water surface at the bottom of the Baltic Sea, there is the bottom weight working as an anchor which provides a first recovery line. 35 meter higher, there is the acoustic releaser equipped with a second 400 meter long recovery line coiled on a roll. The acoustic releaser is used for regular recovery. If the acoustic release mechanism does not work, the bottom weight's recovery line can be used for an emergency retrieval. Another ten meters higher you can find the underwater winch equipped with floatation panels. It is powered by four batteries and enables taring up and down inside

the water column. As the last component, the PIP is situated on top of this construction. It contains the sensor system and has its own floatation panels. There were used two slightly different profiling platforms. From 2010 till 2013 the PIP-1 has been used. In 2013, it was replaced by another platform called PIP-2.

Sensors typically output their measurements at regular intervals. [VVM12] Concerning this sampling station it is important to notice that there is no steady investigation, since it is deployed only during determined time frames. One deployment is defined as the release of the sampling station as well as the time between release and recovery of the station. The deployments are consecutively numbered which means that each measurement cycle has a distinct number as identifier. Dependent on the programmed configuration the deployment interval ranges from several weeks to several months. There are measured various so called profiles during one single deployment. The number of profiles thereby also depends on the programmed configuration. A profile consists of two phases: First, the PIP moves up until it is right below the surface. Then, the PIP moves down again till it arrives at the underwater winch. The execution of both consecutive steps is defined as one profile.

The profiles are programmed by the researchers to be performed at a fixed time. At this specific point in time, the underwater winch releases the lock for unwinding the line. Simultaneously, the floatation panels of the PIP ensure that the line unwinds in control. In the best case, gathering the sensor data by the PIP should be synchronized with the unlocking mechanism inside the underwater winch. Since there is no communication between the PIP and the underwater winch, internal real-time clocks are used to ensure this synchronisation. Because those clocks offer a slightly lower precision than direct communication, there is a little difference in time, the so called drift, between the two components. As a consequence, the PIP starts gathering sensor data two minutes before the underwater winch actually releases the lock.

The station is equipped with several different sensors developed to measure different parameters. This mainly includes a Sea & Sun Technology CTD 90 M Multiparameter Probe, a Rinko Oxygen Optode, a TriOS ProPS Sensor and an Aquadopp Flow Meter from Nortek. Like the name suggests, the CTD Probe measures conductivity, temperature, depth (CTD) and additional parameters like pressure or light transmission. Moreover, the Rinko Optode is used to measure oxygen content. In addition, the TriOS Sensor measures nitrate values based on spectral data and the Aquadopp Sensor investigates parameters used for describing the water flow. All sensors are connected to a central data logger, the IOW PowerLogger. It creates three seperate txt-files for the CTD-meter, the spectral data sensor and the flow meter. Thereby, few data from the flow meter and the spectral data meter is also written in the CTD-file. In contrast, data from the oxygen optode and the CTD-meter is included solely in the CTD-file. In summary, the sensor system investigates a wide range of water parameters including CTD (conductivity, temperature and depth), soluted oxygen, turbidity, Chl a florescense, pH value and redox potential and stores the measured data in three separate files.

2.2. Gathered Information and Data Properties

As mentioned earlier, the data-files are generated by the IOW PowerLogger which produces three kinds of data files that are saved in txt-format. The files are generated per day and can contain information of more than one profile. The data has no standardised structure like XML or JSON and consists of either key-value-pairs or single values separated by delimiters such as commas or spaces. In addition, each file contains specific metadata describing the file or characterising the format of the payload. This produces three divergent structured TXT-files. Since there indeed is a certain pattern of structure in

each document, but this structure is dependent on the relative sensor and differs from deployment to deployment, the files will be classified as semi-structured documents.

Files generated for the CTD-meter are saved in the AyymmddX.txt format. The first letter has no special meaning anymore. It was once added to the file name to ensure that previous loading processes in Matlab work unproblematically. The letter is followed by a date and another letter "X", which will always be an A and is provided by the IOW PowerLogger. The CTD-files contain the majority of the measured data. Like shown in Listing 2.1, the metadata is registered to begin with an @ symbol. In contrast, the payload begins with a predefined string and a comma as separator. It offers information about date, time, seconds, milliseconds, battery voltage, pressure, temperature and conductivity. In addition, there are further variables like pH-value which are considered to be non-essentiell for the Matlab analysis. Moreover, it can happen, that some values appear to be redundant because they are measured doubly. In this case, they are reduced to one record within the analysis.

```
...
@NAME,DATE,TIME,FRAC,SEC,VBatt,Press,Temp,Cond ...
@UNIT,YY-MM-DD,HH:MM:SS,MS,RUNSEC,V,dBar,Ã¸C,mS/cm ...
...
@METAEND
|-----------------------------------------
@DATASTART "2015-05-13" "18:01:19"
$PSDA1,2015-05-13,18:01:19,093,10,33295,62089.000,49171.000,15590.000,14307.000 ...
$PSDA1,2015-05-13,18:01:20,544,11,33301,62089.000,49093.000,15590.000,14305.000 ...
$PSDA1,2015-05-13,18:01:21,015,11,33303,62093.000,49067.000,15589.000,14302.000 ...
...
```

Listing 2.1: Extract of an exemplary CTD-File

The TriOS-files are saved in the Pxddhhmm.X.txt format. The "P" originates from ProPS which is the name of the sensor. The next letter "x" can either be a C for calibration or an M for measurement. This is because the spectral data meter measures either with light turned off (C) to investigate the dark current of certain photo detector canals or with light turned on (M). In actual fact, there is no difference between C and M, since the light technically is turned on all the time. Finally, the first two letters are followed by a date and another letter "X" which is defined as an consecutively counting index variable provided by the PowerLogger. As you can see in Listing 2.2, the data consists of several segments separated by identifiers in square brackets. It contains metadata such as spectrum and attributes as well as payload that splits into four columns. The first column shows the pixel number of the used photo diode. The second column is defined as a 16-bit value being proportional to the measured illuminance. The last two columns are not relevant and represent a mistake counter and the mistake status.

```
...
[DATA]
0 9 0 0
1 1357 0 0
2 1327 0 0
3 1330 0 0
...
```

Listing 2.2: Extract of an exemplary TriOS-File

The names of the Nortek-files are structured in the NOyymmdd.txt format where "NO" represents an acronym for the fabricator Nortek followed by the date. As Listing 2.3 illustrates, the files contain no metadata beside date and time and the measured information is encoded as hexadecimal values. Thus,

it is impossible to read from the files before the data is converted into human-readable values.

```
2015-04-21 06:47:30 A521C400263610160201000000007300F23AA400F7F ...
2015-04-21 06:47:32 A521C400263710160201000000007300F23A7C00F9F ...
2015-04-21 06:47:33 A521C400263810160201000000007300F13A9B00EDF ...
...
```

Listing 2.3: Extract of an exemplary Nortek-File

Additionally and not generated automatically by the PowerLogger, there are metadata-files generated manually by the researches. There is one XML-file per deployment containing data about deployment-specific calibration. It is saved in as a deployment_xx.xml where "xx" represents the unique deployment number. In sum, the metadata file contains data about a wide range different sensors including FracSec - a sensor for time, PressureSensor - a sensor for subaqueous pressure, TempSensor - a sensor for temperature and ConductivitySensor - a sensor for conductivity. Listing 2.4 exemplarily shows the calibration data for the conductivity sensor.

```
<ConductivitySensor>
        <SerialNumberCTD>468</SerialNumberCTD>
        <ChannelNumber>10</ChannelNumber>
        <SerialNumberSensor>4681</SerialNumberSensor>
        <CalibrationDate>16.07.2014</CalibrationDate>
        <A0>-8.86499939e-002</A0>
        <A1>1.01438668e-003</A1>
        <A2>9.35514578e-012</A2>
        <A3>1.10249781e-016</A3>
</ConductivitySensor>
```

Listing 2.4: Calibrated Metadata for Measuring Conductivity

In terms of sensor metadata, each XML-file includes the serial number of the CTD-meter, the channel number, the serial number of the sensor and the calibration date. Moreover, it does contain four different calibration values $(A0, A1, A2, A3)$ and a polynomial coefficient x that is defined as a positive integer between zero and 65535 because actual measures are digitized by a 16-bit converter. The sensors are using those values for calculating SI-units by polynomial evaluation which depends on the calibrated values. The SI-units are then added to the CTD-file. By slighty pre-empting the Matlab scripts described in section 2.3, the following equation illustrates how conductivity C can be computed by using the polynomial coefficient and the generated calibration values.

$$C = A0 + x * (A1 + x * (A2 + x * A3)))$$

2.3. Performed Matlab-Analysis

To process the data saved in the three files, the IOW uses Matlab. It was developed by The MathWorks Incorporation in 1984 and is considered to be a mathematical programming language used for data analytics.[Ord13] It is not database-supported and is optimized for solving engineering and scientific problems on a file-based level. According to the documentation it is used for machine learning signal processing, image processing, computer vision, communications, computational finance, control design or other and the key features split into high-level language for scientific and engineering computing, desktop environment tuned for iterative exploration, graphics for visualizing data, apps for curve fitting, data classification,

signal analysis or control system tuning, add-on toolboxes for a wide range of engineering and scientific applications, tools for building applications with custom user interfaces, royalty-free deployment options for sharing Matlab programs with end users and interfaces to a wide range of other languages including C/C++, Java, .NET, Python, SQL, Hadoop and Microsoft Excel. [Mat19]

Regarding the GODESS, Matlab is used for several purposes but mainly for processing and visualizing the data. The aim is to generate graphic profiles showing correlations between two or more parameters or time-dependencies of the measured variables. Thus, researches investigate the development of certain parameters over time. [FWW+10] The objectives are to detect divergences between different profiles, abnormalities or to check the data for data leaps in order to optimize the instrument's calibration or to find conspicuous patterns and trends. For all analysis types there are used three Matlab scripts. Thereby, most of the analysis is performed by one single script containing 1355 lines of code. It is separated into several sections which can be executed independently or as one unit. This illustration following the original Matlab script code shows which steps are generally involved in the process:

0. Read Metadata
1. Read raw CTD-Files
2. Merge Split Casts
3. Merge Files
4. Interpolation
5. Read Nortek-Files
6. Process Nortek Data
7. Read Seabird Data
8. Comparison of CTD Data
9. Read TriOS-Files
10. Process TriOS Data
11. Plot Timeline
12. Deconvolution

As a first step (Section 1), the metadata is loaded which basically involves getting the meta-information from the XML-files. Secondly (Section 2), also the raw CTD-files are read in. In a next step (Section 3), split casts are merged. Since some casts can be split over two different days the data has to be modified. This includes resorting the data if a profile exceeds the current date. Next, the different CTD-files are merged (Section 3). The data first gets unified and converted from their saved data format. Thereby, sensor data gets converted into SI-units using the coefficients from the XML-file. As a result of this section, the information from all files is merged together. Subsequently, the CTD data is interpolated to sigma coordinates and thereby to a constant pressure axis (Section 4). Following this, the data is plotted for the first time.

As a next step (Section 5), the Nortek-files are loaded and then processed. Thereby, the Nortek data is sorted and inserted into the CTD data (Section 6). The next step (Section 7) includes loading data that is not gathered by the sampling station. In this section data from the Seabird is read in which is information gathered by a research ship that is saved as an CNV-file. After that, CTD data gathered from the Ship is compared with the CTD data measured by the GODESS' sensors (Section 8). Finally, the TriOS-files are loaded and it comes to plotting the data for a second time in the next stage

(Section 9). Firstly, the data is interpolated again to constant pressure axis, then it is plotted (Section 10). Consequently, a plot timeline can be visualized involving data from several deployments (Section 11). As a last step (Section 12), the slow sensor data such as data about oxygen and pH-value is deconvoluted.

Figure 2 illustrates a typical output produced by the second last section of the script. It shows the correlation between measured temperature T and sea depth as a function of time for multiple deployments. Thereby, the temperature is measured in degrees Celsius and the depth is measured in meters. The right scale of the plot shows a colored spectrum from warm red to cold blue to illustrate the differences in temperature. Generally, six deployments are involved. By plotting such graphs, the researchers hope to detect some trends or anomalies inside the data. Looking at this graph for example, we clearly see some anomalies between August and December at a depth between 30 and 60 meters. Furthermore, the plot demonstrates that the data is not measured steadily, since there are some gaps between the different sub-plots. Furthermore, there are white spaces inside some graph sections. This indicates that at some point for some reason data could not be collected. Hence, there can be possibly some leaps inside the data of a deployment.

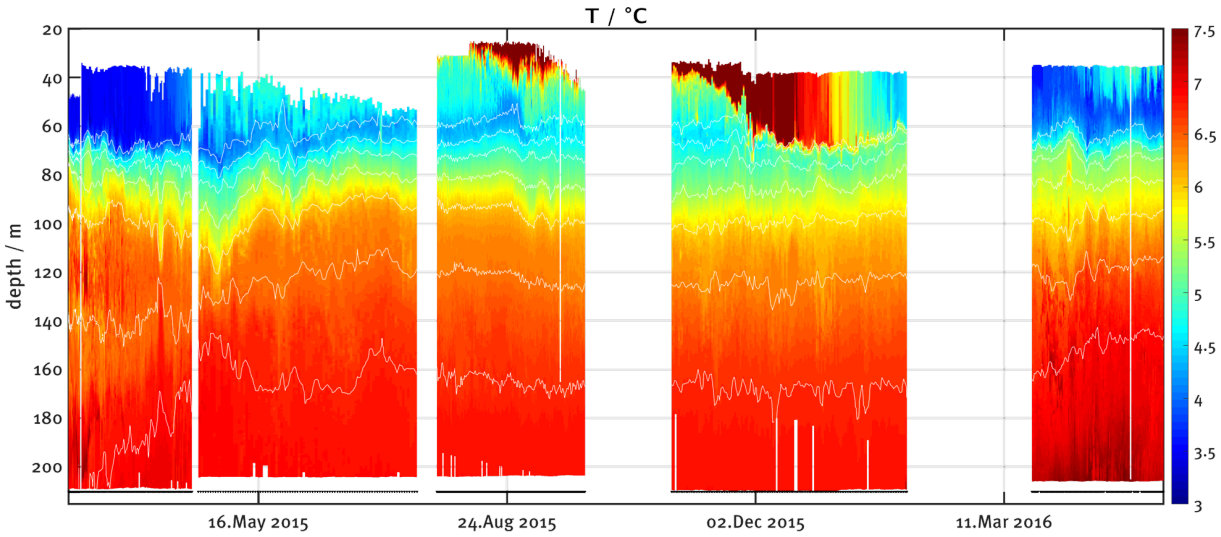

Figure 2.: Exemplary Plot across multiple Deployments (Temperature) [Pri19]

There are two additional Matlab scripts used for calculating practical and absolute salinity. Together, they contain further 283 lines of codes - 194 lines for computing practical salinity and 79 lines for computing absolute salinity. Whereas practical salinity is calculated directly from the conductivity of the sea water, absolute salinity is based on the density of the sea water. Thereby, absolute salinity is defined as the mass fraction of dissolved material in sea water [ISL10] Each of both salinity scripts relies on a certain toolbox from the Matlab universe.

The practical salinity $S(C,T,p)$ is dependent from conductivity C, temperature T and pressure p and is calculated using the toolbox $gsw\_SP\_from\_C(C,T,p)$. This toolbox basically uses the so called PSS-78 (Practical Salinity Scale) algorithm developed in 1978 and later extended by Kenneth D. Hill in 1986. [HDW86] The PSS-78 algrithm is only valid in the range of $2 < SP < 42$. If the PSS-78 algorithm produces practical salinity less than two, the practical salinity needs to be recalculated with a modified formula developed by Hill to ensure that it is exactly consistent. [MB19] Subsequently, absolute salinity $SA(S,lon,lat)$ is calculated based on practical salinity S, longitude lon and latitude lat using the Matlab toolbox $gsw\_SA\_from\_SP\_Baltic(S,lon,lat)$. Since practical salinity is non-negative by definition, this function changes any negative input values to zero. [DJB19] In addition, this program is highly specific and will only produce absolute salinity values for the Baltic Sea. Figure 3 demonstrates how salinity can

later be plotted. As an example, it shows the correlation between absolute salinity and density ρ as a function of time for multiple deployments between 2015 and 2016. Similarly to the temperature graph, there are some anomalies and leaps inside the data.

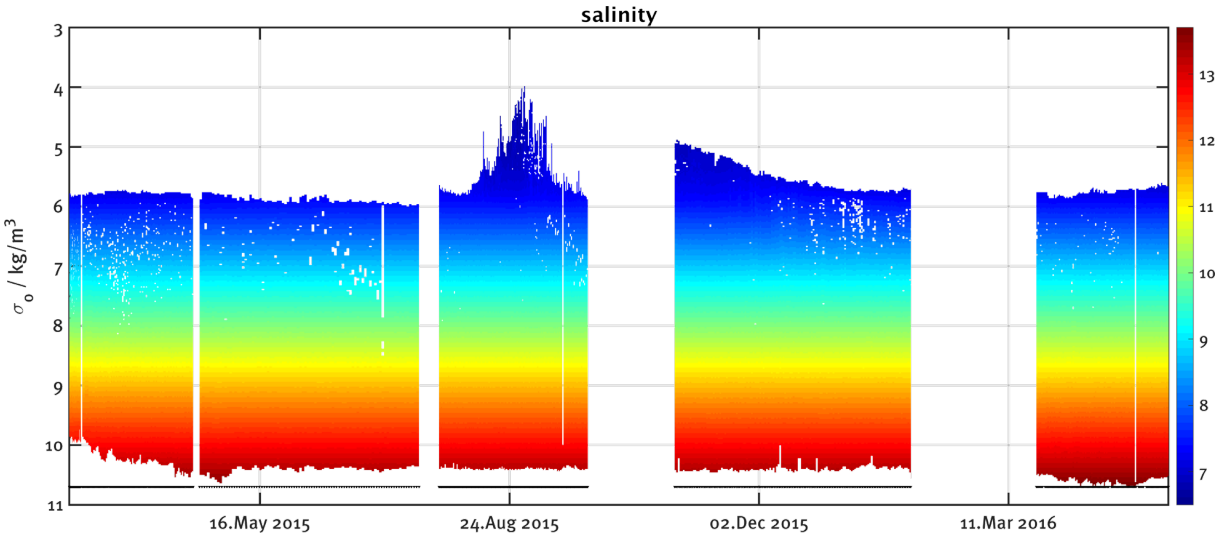

Figure 3.: Exemplary Plot across multiple Deployments (Salinity) [Pri19]

Analyzing the Matlab scripts in terms of its content, multiple functions and operations can be identified. Surely, there are basic operations like reading, writing, updating and exporting data of different types. Furthermore, there can be found a wide range of functionalities. Particularly in the main script, there are various complex case differentiations and iterations as well as sort and merge algorithms continuously used for processing and analyzing the data. Since Matlab is developed mainly for dealing with matrices, the analysis contains various vector operations including element wise and matrix multiplication. As mentioned before, polynomial evaluation is used for converting parameters from their hexadecimal values into human-readable data. Furthermore, the researchers somehow query ranges while searching for leaps and abnormalities. On top of that, some complex statistical techniques are applied. For instance, to compare different CTD casts, variance analysis is used. Moreover and in terms of calculating intensity counts, the researchers even rely on multiple linear regression. Another key functionality is the plotting of various types of graphs involving data from two or more parameters.

For calculating practical and absolute salinity there are also used several consecutive functions. The estimation of PSS involves case differentiation as well as matrix multiplication, but mainly relies on complex polynomial functions. Absolute salinity then is calculated using a simple case differentiation involving highly complex matrix operations and interpolations.

In summary, ten following functionalities can be identified. This involves complex case differentiation, iteration, sorting, merging, matrix multiplication, range querying, polynomial evaluation, variance analysis, multiple linear regression and plotting graphs. Together, they constitute a wide range of functionalities to be potentially transformed to database-supported evaluations. As some users prioritize SQL-based systems for performing data analytics and others prefer NoSQL databases, this thesis will transform some of the listed functionalities to both SQL and NoSQL platforms. In order to comprehend the differences between these platforms and to find potential database systems meeting the requirements of the GODESS data, the next chapter will explain how these platforms are characterized.

3. State of the Art

Since statistical, numerical and mathematical languages purposely provide proper analytical performance when it comes to specially tailored applications [Ord13] but in such a case lack in terms of storage usage, multi-user operations, data provenance, change management and privacy aspects, it seems obvious that an increasing number of companies and researchers rely on database-supported solutions. As mentioned earlier in this paper, the IOW likewise considers a digital change. Therefore, the following chapter provides a broad overview about intended database properties, relational database systems such as SQL-based systems and the current development of NoSQL languages. Furthermore, features, advantages and disadvantages as well as application areas for SQL and NoSQL systems will be discussed. Thereby, the discussion starts by describing two of the most common principles for database properties in section 3.1. After that, section 3.2 will explore how SQL-based database systems are characterized. Finally, section 3.3 will differentiate from SQL-based systems by depicting the characteristics of NoSQL database systems.

3.1. ACID vs. BASE

Although SQL and NoSQL databases strongly differ from each other, both are considered to be database management systems. A database management system thereby will be defined as any system that stores and administers data and performs transactions to process this data. There are several different types of database systems, some relying on relations or tables and some not. In the scope of the set goals, this thesis will focus on the differences between SQL-based databases and NoSQL systems. No matter what type of database is considered, it is always important to find a proper conceptual representation of the data structures that are required by a database. This representation is called **data model** and the data structures include data objects, the associations between these objects and the rules which govern those operations. [KR13] Dependent on the purpose, a database management system can have different designs and different designs require individual properties. Two of the most common principles defining those properties are **ACID** and **BASE**.

CAP-Theorem

Whereas SQL-databases rely on ACID principles, most NoSQL solutions apply BASE principles. It is important to notice that both principles are based on the so called **CAP-Theorem** which is illustrated in Figure 4. This theorem basically suggests that there are three properties a database potentially can fufill - **C**onsistency, **A**vailability and **P**artition tolerance . A database is considered to work consistently, if every read receives the most recent write or an error. [Bre00] This implies that the data is always the same in every replication on every server. [Web10] Availability is satisfied, if every request receives a response without having the guarantee that it contains the most recent write. [Bre00] Thus, data must always be accessible. [Web10] The last property, partition tolerance, is fulfilled, if the system continues to operate despite an arbirary number of messages being dropped by the network between nodes. [Bre00] Hence, the database needs to work fine despite network and machine failures. [Web10]

Figure 4.: CAP-Theorem

The CAP-theorem postulates that you can have at most two of these properties for any shared data system. [Bre00]. Accordingly, only two of the three different aspects of scaling out can be achieved fully at the same time. [MH13] Following this assumption you can either fulfill availability and consistency at the cost of partition tolerance, consistency and partition tolerance at the cost of availability or partition tolerance and availability at the cost of consistency.

ACID

As mentioned before, both ACID and BASE are derived from the CAP-theorem. ACID is mainly used as a set of rules for relational database transactions to ensure consistency. [AB13] A transaction can be described as an operation processing data that is started by release and ended by commit. [MH13] Most of classical database systems are based on transactions to guarantee the integrity of data. ACID is an acronym for **A**tomicity, **C**onsistency, **I**solation and **D**urability. Atomicity means that a transaction is completed when all operations are completed and that otherwise a rollback is performed. [AB13] In terms of consistency, the ACID principles imply that the database ensures integrity of data before and after each transaction. [AB13] Furthermore, a database relying on ACID principles is isolated, if all transactions are independent and cannot affect each other. [AB13] Finally, Durability is fulfilled, if the system guarantees that the results survive any subsequent malfunctions once a transaction is committed. [HR83].

BASE

The BASE principles still follow the CAP-theorem, but prioritize different properties. For a growing number of applications and use cases, availability and partition tolerance are becoming more important than strict consistency which is preferred by ACID proponents. [SK11] Even in distributed systems, two of the three qualities must be chosen. [AB13] Those systems that obviously need to ensure partition tolerance over several clusters, prefer availability over consistency. [SK11] Thus, BASE stands for **B**asic **A**vailable, **S**oft-state and **E**ventually consistent. It is intended that the consistency after a transaction is not a solid state anymore, but a soft state instead and it shall be reached not right after finishing the transaction, but rather sometime. [Web10] Hence, BASE forfeits the ACID properties of consistency and isolation in favor of availability, a graceful degradation and a faster performance. [SK11] Most NoSQL databases have loosened up the requirements on consistency [MH13] in order to achieve almost permanent availability. [Web10] This includes constraints on the classical data model to enable better partition schemata. [MH13]

3.2. SQL-Database-Systems

The main purpose for constructing databases is to query the valuable data stored inside. Therefore, query languages are used which can be specified as computer languages used to manipulate data inside a database.[NPP13] Since 1970 data stores have been relying on the traditional calculus of Relational Database Management Systems (RDBMS) providing comprehensive ad hoc querying facilities by Structured Query Language (SQL).[Cod70] Although there are exceptions, relational databases can be generally considered to be one single SQL DBMS product.[VFKV16] While implementing RDBMS a lot of attention is paid on developing conceptual, logical and physical data models.[KR13] The data is stored in tables, so called relations, and the values are stored row-wise.[Ord13] The model is mainly dedicated to structured data and to handle transactions respecting the ACID properties.[OBLB15] Thereby, SQL represents the most commonly used query language by relational databases.[NPP13] A typical example of a SQL query will be illustrated in section 4.2.

Features and Benefits

Above all, SQL databases ensure data integrity and transaction consistency.[OBLB15] Moreover, it is possible to highlight relationships between different data sets and even relationships between a single metadata set and several other databases.[Möl16] They offer advanced functionality to manage, update and query data[OBLB15] and can efficiently process transactions and SQL queries on data.[Ord13] Hence, RDBMS are a good fit for small but frequent read and write transactions and for large batch transaction with rare write accesses.[TB11] Furthermore, SQL queries allow processing the data in-situ as well as relational query optimization including forcing hash-joins, clustered storage for aggregation, denormalization for avoiding joins, pushing aggregations before joins for compressing tables and balancing row distribution among the processing units.[Ord13] On top of that, privacy issues can be avoided or strongly reduced by using techniques such as data control and isolation.[Möl16]

Additionally, they can be vertically scaled very easily which refers to increasing the capacity and the performance inside a single server.[OBLB15] This kind of scalability is easier to achieve than horizontal scalability[KR13] which is increased by adding additional machines or servers. Even though there exists parallel RDBMS for data warehousing and large-scale analytics and some databases have achieved a certain level of integration between SQL and parallel operations[Ord13], features like sharding and MapReduce were not originally created for those systems.[OBLB15] Nevertheless, they perform well in parallel querying and Online Transactional Processing (OLTP).[Mom01] Nowadays, it is even possible to perform data mining on RDBMS, since most processing on large tables can be done inside the database and only partial processing needs to be done on smaller tables exported to an external mathematical library or statistical program.[Ord13] In addition, it can be assumed that many companies appreciate that relational databases offer commercial support and maintaining.[OBLB15]

Another feature is object-relational extension as **User-Defined Function (UDF)** that does not increase the computation power, but makes programming easier and many optimizations feasible by allowing pushing more mathematical processing into main memory.[Ord13]. Users can use UDFs to implement individual functions using either SQL or another programming language. All in all, SQL-based databases ensure more reliability in comparison to NoSQL database systems.[OBLB15] Although this property makes them suitable for a wide range of applications requiring strong consistency such as management of financial transactions, there are some lacks that unintentionally formed a basis for the simultaneous development of NoSQL systems.

Drawbacks

SQL systems were almost in monopoly position over the last decades and, on top of that, they still represent the widest used and most popular database solutions.[Ord13] Despite the broad variety of mentioned key-features and benefits, there are some drawbacks on RDBMS. Firstly, relational databases primarily perform on single servers.[SK11] Although they scale well vertically, they have issues with scaling out horizontally.[OBLB15] This makes them difficult to expand[HHLD11] and they lack in performance especially for large data sets.[KR13] They turned out to be less efficient when dealing with big data[OBLB15] and therefore cannot compete with parallel systems especially designed for executing MapReduce (see section 3.3 for explanation) to analyze, for instance, web-scale text data.[Ord13] Additionally, they are not suitable for heavy read and write workloads.[TB11]

Due to their rigid relational schema design, it is inconceivable that tuples have different attributes.[Web10] Consequently, they lack in dealing with unstructured data that needs dynamic schemata.[KR13] Furthermore, change management can be very difficult to handle and the row storage model is considered to be less rapid than column-oriented stores.[OBLB15] In addition, there are only a few RDBMS that are open-source like for instance PostgreSQL or MySQL. In concerns of Online Analytical Processing (OLAP), there can be the problem of integrating statistical and machine learning methods, since RDBMS lack of the comprehensive set of techniques already available in mathematical languages like Matlab, statistical languages like R or numerical languages like LAPACK.[Ord13] Subsequently, it is difficult for users and researchers to incorporate new algorithms or change existing ones and data structures, incremental computations and matrix operations are tricky to program.[OBLB15] As mentioned by Ordonez, it is quite common for users to export data sets to a statistical software or a parallel system.[Ord13] He elaborates on this by mentioning that most of the querying is done inside the RDBMS, but the computation of statistical models and pattern search is more commonly performed outside.

3.3. NoSQL-Database-Systems

NoSQL databases have put doubts on the usefulness of data models trying to eliminate the need for data modelling.[KR13] Advocates believe that the current "One size fits it all"-thinking is wrong and therefore other systems are required.[SK11] The term NoSQL means "**N**ot **o**nly SQL" and was first introduced by Carlo Strozzi in 1998 for his open source relational database that did not offer an SQL interface as a general hypernym for relational databases that do not use SQL.[Str98] In 2009, the term was reintroduced by Eric Evans at the event "no:sql(east)"[KR13] which was a conference about non-relational databases in San Francisco.[OBLB15] Indeed, some experts are using the term with the meaning of a completely non-relational system and there even exist some middle-ware appliances between relational and non-relational databases.[TB11]

The Movement of NoSQL

According to Strauch, the search for alternatives can be explained by two trends: (1) The continuous growth of data volumes to be stored and (2) The growing need to process larger amounts of data in shorter time.[SK11] The main reason for this development is that Web 2.0 increased the use of data quantity stored in databases all over the world[AB13], since users were not restricted anymore to only read information, but furthermore contributed to the online data pool by writing information inside the databases. Additionally, the cost of posting and exchanging information became cheaper after 1990 due to the growing popularity of the Hypertext Transfer Protocol (HTTP) which led to a flood of information on the internet.[KR13] Another factor was the proliferation of clouds, especially in the sense of Database as a Service (DBaaS).[OBLB15] DBaaS can be defined as a set of tools providing final users with seamless mechanisms

for creating, sorting, accessing and managing their proper databases on remote servers which makes them the most appropriate computational data framework for implementing big data repositories. [CSD11] The major requirements of cloud computing split into high until almost ultimate scalability and a low administration overhead. [SK11] The excessive generation of information by modern applications, cloud computing and smart devices [VFKV16] finally led to a phenomenon called Big Data.

Das and Kumar provide an example that perfectly describes this evolution. While it took from the dawn of the civilization to 2003 to create five exabytes of information, we now create the same volume in just two days. [DK13] The Big Data characteristics split into four challenges:

- Data **V**elocity
- Data **V**ariety
- Data **V**olume
- Data **V**eracity

Data velocity refers to rapidly and continuously updated data streams from different sources and locations. [VFKV16] Thus, data is generated in real-time with demands for usable information to be served up as needed. [DK13] Moreover, structured, semi-structured and even unstructured data must be stored which leads to a massive data variety. [VFKV16] On top of that, the data is collected from new sources that have not been mined for information in the past. [DK13] Data volume means that it became common to deal with a huge number of data sets with sized of several terabytes or petabytes. [VFKV16] Particularly, the growth of unstructured data contributes to this issue. This data is strongly heterogeneous, variable in nature, comes in many forms and is growing faster than structured data. [DK13] Finally, data veracity, which was classified as an additional challenge just a little while ago, states that data quality has a massive impact on how successfully Big Data can be used. [VFKV16]

NoSQL databases arose alongside major intenet companies such as Google, Amazon and Facebook which had challanges in dealing with these huge quantities of unstructured data with conventional RDBMS. [MH13] The first NoSQL database was Google's BigTable which was introduced in 2006. Only one year later, Amazon followed by releasing their database called Dynamo. [KR13] Another instance of the most significant application scenarios where Big Data arise is scientific computing. [CSD11] All together, there is a bunch of new priorities being inevitable for those companies and the scientific world. Firstly, there is the importance of information promptness. [AB13] Secondly, there is a need for technologies requiring only low administration overhead and high scalability. [OBLB15] Moreover, there is an increased need for real-time analysis and most information saved in companies' repositories is stored as unstructured data models. [DK13] Thus, new techniques were developed allowing Google to index the web, Facebook to build social graphs and Netflix to recommend movies. [Los13] Nowadays, there are over 120 of different NoSQL dabases [TB11] and many universities have started teaching about these data stores as a part of their curriculum. [KR13]

Features

NoSQL databases differ from SQL systems in terms of features and properties. According to Weber [Web10], the NoSQL archive defines them as non-relational, distributed, open-source and horizontally scalable. Thereby non-relational means that the database is not based on a relational model anymore and therefore has no relations or tables. Distribution implies that the data is stored on and managed by different machines or servers. Moreover, a database is considered to be open-source, if everyone can look into the source code for free or even change or compile it. Finally, he elaborates on horizontal scalability

which can be explained as a positive, nearly linear correlation between the number of servers you add to the distributed system and the performance level of the whole cluster. Another definition provided by Tudorica and Bucur describes NoSQL databases as database systems which are distributed, may not require fixed schemas, usually avoid join operations, typically scales horizontally, does not expose a SQL interface and may be open-source. Weber also mentions that there is a selection of properties that can be potentially added to this quartet [Web10], but since no final definition of such database systems is yet applied, the prior illustrated definitions deliver a sufficient imagination.

Although NoSQL databases are classified within the spectrum from ACID to BASE [Web10], most systems respect BASE properties. [OBLB15] Not least because implementing ACID into distributed systems is not trivial, since the major problem is to ensure consistency of data sets being distributed over several servers. [Web10] Due to the flexible architecture of distributed systems [OBLB15], the schema is not fixed [VFKV16] and databases are not primarily built on tables anymore. [MH13] Furthermore, the data is processed massively parallel across a large number of commodity servers. [MH13] Thereby, the same file is stored on different servers as instances of a resource which is commonly known as replication. [Web10] Most NoSQL database systems even provide sharding which involves cutting up the database into multiple tables to run them on large clusters or grids. [SK11] Since each NoSQL database has developed its unique way to manage, extract and query data [OBLB15], they do not use SQL for data manipulation. [MH13] Hence, querying this new generation of databases is strongly data-model specific as each database was developed for a specific purpose and therefore relies on an individual data-model. [KR13] Apart from that, distinction between data definition language and data manipulation language like in SQL systems does not exist in NoSQL databases. [VFKV16]

The input data is usually large and the computations sometimes have to be distributed across hundreds or thousands of machines in order to finish the work in a reasonable amount of time. [DG04] Therefore, most NoSQL databases provide parallel processing relying on a principle that was already mentioned earlier and is called *MapReduce*. This technique automatically distributes analytic operations across a cluster of computers and can work on top of a distributed file system providing great flexibility, efficiency, expandability and low cost. [Ord13] The underlying runtime system automatically parallelizes the computations, schedules inter-machine communication and even handles machine failures. [DG04] Thereby the programmer specifies two basic functions: *Map*() and *Reduce*(). Figure 5 illustrates how MapReduce works. Basically, *Map*() partitions computational tasks into smaller computational tasks and assigns them to then appropriate key-value-pairs. [CSD11] Written by the user, it takes a pair as input [DG04] These smaller tasks are executed efficiently by exploiting parallelism. [CSD11] In a next step, the MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function. [DG04] Thus, the final result of *Map*() is obtained via the *Reduce*() operation that combines all the values sharing the same key. [CSD11] Finally, *Reduce*() which is also written by the user accepts an intermediate key and a set of values referring to this key and merges the values together. [DG04]

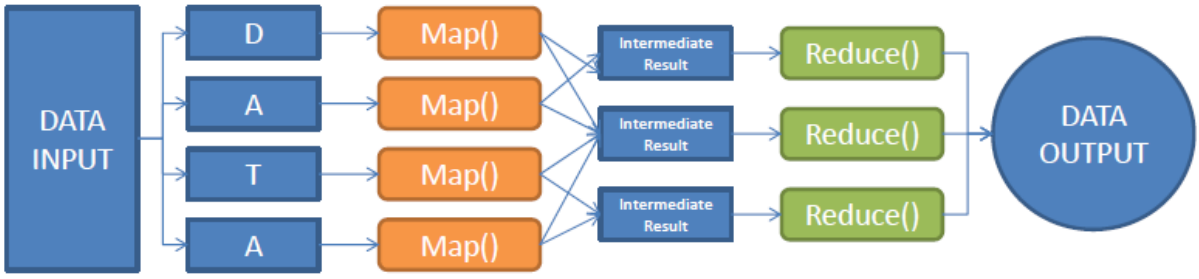

Figure 5.: Illustration of a MapReduce Process

Benefits

Overcoming the restrictions of previous SQL-based systems, NoSQL offer a wide variety of advantages. Firstly, there is a broad range of models to choose from [NPP13] which delivers a proper solution for every specific case of application. Hence, they are considered to be serious competitors due mainly to their data model flexibility [FC13] which allows to store data without pre-defining a schema. [OBLB15]. Often, a database administrator is not required [NPP13] and NoSQL systems turned out being able to resolve the storage problems of massively unstructured data sets. [OBLB15] Moreover, such systems provide easy change management [OBLB15], since it is possible to change the data model at any time. [OBLB15]. They require lower all around management and have lower computational costs. [HHLD11] Most of them are open-source [SK11] which makes them easily affordable and creates a high level of transparency. As a consequence, users do not need to case about licensing and commercial support issues and can simply service themselves. Furthermore, NoSQL systems provide a significantly higher data throughput than traditional DBMS [SK11] and offer high concurrency processing of reading and writing operations with low latency. [HHLD11]

Beside these factors horizontal scalability remains the main advantage of NoSQL services. Large amounts of computers can be connected creating a cluster and its performance exceeds a single node unit equipped with additional processors and memory. [AB13] This kind of scalability is cheaper to achieve [OBLB15], since it is more inexpensive to have a large amount of computers with fewer resources than buliding a supercomputer. [AB13] As one consequence of scaling out horizontally, distributed NoSQL systems do not rely on highly available hardware. [SK11] In addition, Oussous et al. [OBLB15] mention that sharding allows to balance the load and to ensure parallel storage as well as processing of data.They expand on this by stating that sharding creates valuable options to add or remove servers from the data layer without affecting the application performance which makes them suitable for handling parallel computations and mathematical equations on large distributed data sets. Moreover, most databases have automated data replication for ensuring a higher fault-tolerance. Hence, the community agrees that processing the data where it resides is faster and more efficient than first transporting it to a centralized system. [DK13]

All in all, NoSQL databases can be specialized enormously well and offer a big number of features for almost every application area. For instance, they seem to be the better option for business situations requiring simplicity, adaptability, high performance analytics and distributed scalability over large amounts of data. [VFKV16] Thus, Moniruzzaman and Hossain [MH13] state that they are suited to exploratory and predictive analytics on semi-structured data. They explore that the relatively inexpensive costs for large volume data storage provide adequate conditions for storing small-packet historical data gathered from logs, call-data records, meter readings and ticker snapshots. Furthermore, they elaborate on other applications by mentioning that the distributed framework makes them ideal for massive batch data processing including aggregations, filtering, sorting and programmatic or statistical algorithmic crunching.

Apart from that, NoSQL languages allow polyglot persistence which means that different classes of NoSQL databases can be used simultaneously within one application. [KR13] Another aspect is that some NoSQL DBaaS providers offer so called ETL-style data transformation. [MH13] Hence, they apply advanced extraction-transformation-loading (ETL) processes that transform raw data to somewhat structured information. [CSD11]. Additionally, NoSQL databases are recommended when mobile applications manage huge amounts of data on a central server. Apart from mobile phones, NoSQL database systems come in many forms and schemas, are flexible and therefore form a basis for many applications. As the community suggests, this mainly includes big data analytics and parallel processing of large-scale data generated either by small devices like logs and sensors or web-applications such as social networks.

Drawbacks

Critics claim that NoSQL is only a hype and some people see such systems as being nothing new, since similar attempts like object databases have been around for decades. [SK11] Indeed, there are some downsides on these schema free designs. First of all, most NoSQL databases are young and immature [HHLD11] and there is a lack of familiarity and limited expertise. [VFKV16] There is no standard query and manipulation language or a standard interface. [OBLB15] Therefore, researchers usually must be a skilled statistician working in tandem with a skilled programmer [MH13] and even data scientists face the challenge to understand the query language of each NoSQL database. [OBLB15] Therefore, it is difficult for the user to switch from one NoSQL database provider to another. [NPP13] Additionally, there exist some security issues, since security is not a subject of priority anymore. Thus, many NoSQL systems do not secure clients properly and server communications do not provide authentication nor auditing mechanisms, because encryption of very large unstructured data sources is difficult to achieve. [OBLB15] This limits the privacy of users or people that are potentially involved analysis targets.

Apart from privacy issues, there are two additional limitations regarding the user-friendliness. Firstly, there is no commercial support which can scare business people in the case of failures with nobody to blame for. [SK11] Secondly, NoSQL databases can be strongly challenging to install and maintain. [VFKV16] Although some providers meanwhile offer install wizards, the majority of NoSQL databases require complex server setups. Other drawbacks of NoSQL systems are that they lack in terms of performance when it comes to complex queries and joins are difficult to achieve. [OBLB15] It can be quite complicated to export data from distributed to undistributed systems [VFKV16] and they lack in dealing with transactions. [HHLD11] Moreover, many statistical functions are too holistic to be distributed over a MapReduce system and MapReduce requires expensive effort and manually programming. Finally, it can surely be an issue that NoSQL databases scale up horizontally at the cost of consistency. [VFKV16] For applications requiring high reliability such as those dealing with financial transactions, NoSQL is only little suitable.

3.4. Sub-Categories of NoSQL Database-Systems

Due to the flexible schema design, there is no standard NoSQL model. Rather we find a broad diversity of different systems that are difficult to cover in a proper classification. Therefore, the community suggests different categorizations and products which are in the same category in one taxonomy are listed in separate categories in another one. [TB11] For analyzing which type of NoSQL system could meet the requirements of the GODESS data, it is important to present the different kinds of databases available. Therefore, this paper focuses on the most commonly used classification which separates NoSQL systems into four categories. Thereby, it is important to notice that some databases belong to more than one category. [Web10] Referring this categorization, NoSQL databases are classified in the following subcategories:

- Key-Value Stores
- Document Stores
- Column Stores
- Graph Databases

Next to the mentioned core databases, there are other NoSQL databases which are not considered in this thesis. This includes object-oriented databases, grid and cloud solutions, XML-databases and multi-value

databases. According to Tudorica and Bucur [TB11] they are also called soft NoSQL systems, since most of them being older or newer systems which are not related to any Web 2.0 service but coincidentally share the traits being described as NoSQL characteristics. The authors also mention that some of those databases even offer strong ACID compliance and relational capabilities. Thus, they may be displaced in a list of NoSQL categories.

Key-Value-Stores

Key-Value Stores can be visualized as relational databases having multiple rows and only two columns: key and value. [NPP13] Actually, they rely on a simple data model without relations or structures [Web10] and most databases are inspired by Amazon's Dynamo. [KR13] Other popular examples are LinkedIn's Voldemort, Redis or Riak. They are schemaless and offer a simple API. [NPP13] Each database simply has a key and a corresponding value together as one data set. [Web10] Thus, the data consists of two parts: a string representing the key and the actual data value referring to this key. [NPP13] Obviously, the key should be unique what is realized with hashes. [Web10] In contrast, the value is semantic [HHLD11] and can be of different types like strings, integers, floats or byte arrays. [Web10] Together, key and value create a key-value-pair [NPP13] which is why those databases are called key-value stores. Hence, items are stored as alpha-numeric identifiers with the associated values in simple, standalone hash-tables. [MH13] Some databases do not even care about the data type of the value. In this case, these inputs are called "Blobs" which refers to an arbitrary binary object which the database does not need to interpret. [Web10]

Their simple data model allows the clients to put and request values per key by using a map or dictionary. [SK11] Thus, values are queried according to the key specified. [NPP13] Key-value stores can handle a very large number of records and they support high volumes of state changes per seconds with millions of simultaneous users. [OBLB15] They do not provide any kind of traditional database capabilities which makes it difficult to create custom views of the data. [NPP13] Consequently, users cannot access the data by value which means that most types of selections are not possible and querying a key-value store for extracting all records containing a particular set of values can be very expensive. [OBLB15] These stores omitted rich ad-hoc querying and analytics features including join and aggregate functions [KR13] in order to enable high concurrency, fast look-ups and mass data storage. [NPP13]

This makes them suitable to look-ups for simple or complex values in extremely large data sets [OBLB15] and lightning-fast, highly scalable information retrieval of the values needed. [MH13] Additionally, they are useful for storing the results of analytical algorithms. [OBLB15] Generally, they are adequate for applications where the schema is prone to evolve. [KR13] This makes them applicable for user sessions and shopping carts [NPP13] as well as for quick and efficient data management in distributed systems. [Web10] Moreover, key-value stores offer higher insert and read rates [VFKV16] and faster query execution times [KR13] compared to traditional SQL databases.

Document Stores

Document Stores are considered to represent the next logical step from simple key-value stores to slightly more complex and meaningful data structures. [SK11] Popular examples are MongoDB and Apache's CouchDB. Unlike key-value stores, the value column contains semi-structured data. [MH13] As the name suggests, document databases use entire documents of different types as data sets. The documents are encoded in a standard data format such as XML, JSON or Binary JSON (BSON). [Web10] They may contain multiple key-value pairs, key-array pairs or even nested documents [OBLB15] and in contrast to key-value stores, the database has to know which kind of document is saved. [Web10] Additionally, there is no strict schema the documents have to conform. [SK11] Thus, document stores fulfill the properties of a schemaless database. [VFKV16,NPP13]

Thereby, only the attributes that are really used need to be specified and the user is able to directly edit the data in the document on the server side [Web10] which would be impossible in concerns of key-value stores. Each document is addressed by using a unique key which can be represented as a simple string or a string that refers to an uniform resource identifier or a path. [NPP13] Moreover, each document can contain different fields of any length [VFKV16] and may consists of similar as well as dissimilar data. [NPP13] Apart from that, the documents can have arbitrary structures as well as attributes associated with them. [Web10] Furthermore, they can be grouped together in specific collections. Compared to RDBMS, those collections refer to tables and documents to records. [KR13]

In contrast to key-value stores, both keys and values are fully searchable. [MH13] Thus unlike key-value stores, document stores allow the user to search for data based on the content of the documents and clients can launch queries either by keys, values and even examples, because the encoded documents contain metadata objects as well. [OBLB15] The main advantage of storing data the way it is done in document stores is that object structures of most programming languages can easily be mapped directly into this representation without the necessity of translators. [KR13] To launch queries, users mostly can either rely on a programming API or a query language. [OBLB15]

Obviously, document stores are an ideal fit for storing and managing documents of different types and structures. They are able to store and manage Big Data sized collections of documents such as text documents and mail messages or to save sparse data that would require extensive use of nulls in an RDBMS. [MH13] Hence, document stores are suitable for applications where data has to be stored as a document with special characteristics instead of uniform sized fields and they serve well, when the domain model can be split and partitioned across multiple documents. [NPP13] This makes them an adequate database for content management systems and blog applications [VFKV16] as well as for web applications, storage of semi-structured data or executing dynamic queries [KR13] and DBaaS providers in general. [NPP13]

Column Stores

Column Stores are based on a hybrid approach relying on relational system's declarative characteristics and various key-value store schemata. [OBLB15] Most databases classified as column store are based on Google's BigTable. [KR13] BigTable has been a forerunner in this field, but soon there were developed additional databases such as Apache's Cassandra, Amazon's SimpleDB, Hyptertable and HBase. Even Amazon's Dynamo can be classified as both key-value store and column database. Column stores save and process data by columns instead of rows [SK11]. They have not subverted the traditional store-by-row-principle of relational databases but come along with a different architecture. [OBLB15] They rely on a structure where the data set with its attributes is not stored in one unit like in row-oriented databases but one attribute of a set of data sets is stored in one unit. [Web10] The data is stored by column across several blocks [Ord13] and each row can refer to a different number of columns that are stored. [VFKV16] In addition, there is separate storage for the columns, each containing the primary key or address of each record and the corresponding column value. [Ord13] Thus, column databases can be considered to represent an extension of the key-value architecture with columns [OBLB15] accommodating multiple attributes per key in a distributed data structure. [MH13]

The main advantage over row databases is that not all columns of those rows satisfying a certain condition of a query need to be retrieved which prevents unnecessary disk Input/Output (I/O). [KR13] Thus, access the values of the columns the way it is done in a column store reduces I/O of the system [HHLD11] and even data can be aggregated rapidly with less I/O activity. [NPP13] This enables efficient queries such as

projection of subsets of columns and allows huge numbers of stored columns as well as a sparse nature of data and frequent changes in the schema. [Ord13] Finally, the architecture allows distributed data storage and large-scale batch-oriented data processing like sorting, parsing, conversion and algorithmic crunching. [MH13]

Having its origin in Big Data analytics and business intelligence applications [SK11], column databases are considered best suited for analytical purposes. [KR13] They are ideal to be used for data mining and other analytical applications [SK11,NPP13] including exploratory and predictive analytics performed by expert statisticians and programmers. [MH13] Additionally, they are not only promising in accelerating statistical processing [Ord13], but can be also used in business intelligence to build high-performance application relying on a shared-nothing massively parallel processing architecture. [SK11] Apart from that, applications performing online analytical processing and data warehouses use columns stores, since the needed aggregations can be done very quickly. [Web10]

Graph Databases

Graph Databases focus on relationships between data relying on the graph theory approach. [VFKV16] Neo4j and Apache's Giraph can be categorized as two popular examples of these data stores. Since SQL database-systems and other NoSQL solutions like key-value stores are inefficient when it comes to highly connected data [OBLB15], graph databases overcome this issue by storing the data in form of a graph consisting of nodes and edges. Relationships between objects are illustrated as a connection between two nodes and may be directed. [KR13] In this case, the edges of a directed graph are called arrows. [Web10] Hence, vertices and edges are used to represent the connections between the data. [OBLB15] Thereby, edges can have properties describing the relationship between two nodes. [Web10] Compared to an entity-relationship model which forms the basis for constructing relational databases, a node corresponds to an entity, a property of a node to an attribute and an edge to a relationship between entities. [KR13] For storing the context of vertices and edges, directed adjacency lists are used [Web10] where every node consists of a direct pointer which points to the adjacent node. [KR13] Thus, graph databases have replaced relational tables with structured relational graphs of interconnected key-value pairings and, additionally, can be considered to represent an object-oriented network of nodes similar to object-oriented databases. [MH13]

Despite the fact that many graph databases are ACID compliant [NPP13], these data stores do not require a pre-defined schema [OBLB15] and have the capability of storing huge amounts of interconnected, semi-structured information. [KR13] They are easy to scale horizontally [Web10], allow to query multiple relationships inside large data sets [OBLB15] and provide index free adjacency instead of performing intensive join operations. [KR13] Thereby, queries are expressed as traversals which makes querying faster compared to RDBMS. [NPP13] In addition, these traversals are considered to be cheaper and simpler than traditional query methods. [Web10] Finally, graph databases provide fast all around performance [VFKV16], offer rollback support [NPP13] and allow replications of vertices and edged as well as graph partition. [Web10]

Although this type of NoSQL database-system is not the most efficient solution for updating sets of large volume data, they are suitable to store not only information about objects, but also about relationships existing among them which makes them useful to store, access and analyze the nature for relationships between two or more items. [OBLB15] This makes them ideal for social-networks, bioinformatic applications, content management systems or cloud management services. [VFKV16] Moreover, they are used in location-based services, shortest path applications and for efficient querying of data in a network. [Web10] Additionally, graph databases are a proper solution for generating recommendations or conducting forensic investigations. [MH13] Security and access control illustrate another field of application for them. [NPP13]

3.5. Summary

All in all, there are two major classes of database systems to distinguish - relational SQL-based databases and non-relational NoSQL databases. Thereby, each database system relies on individual consistency models and provides its individual features. Table 1 summarizes the main differences between SQL and NoSQL database systems in concerns of selected criteria. Whereas relational database systems are mostly ACID compliant, NoSQL databases rely on BASE properties in order to achieve higher availability. Thereby, SQL databases use pre-defined and rigid schemata while NoSQL platforms provide dynamic data models. Relational databases mainly use SQL as query language. In contrast, NoSQL databases do not provide a standard query language, since each system provides its own individual querying language.

Moreover, NoSQL databases allow different attributes and try to avoid join-operations in order to achieve higher availability. On top of that, there are more open-source databases in the NoSQL area than in the world of SQL databases. Although, SQL databases provide commercial support and are friendlier to use and to install, they require far more administration overhead. Additionally, SQL systems scale well in the vertical direction and therefore perform better on single servers. On the contrary, NoSQL systems scale better in the horizontal direction and perform better in a distributed node system. RDBMS perform better in terms of OLTP-operations and meanwhile include a growing variety of OLAP-operations, as well. NoSQL databases seem to perform better in automated sharding and MapReduce, although some SQL databases already started to include these parallel processing techniques. Furthermore, there are some RDBMSs relying on distribution and parallel processing since decades and therefore have integrated parallel processing in a more generally manner than NoSQL systems that are providing sharding and MapReduce.

	SQL	**NoSQL**
Data Model	Pre-Defined, Rigid & Relational Schema	Dynamic, Free & Non-Relational Schema
Age	Multiple Decades	Few Years
Dominant Consistency Model	ACID	BASE
CAP-Priority	Consistency	Availability
Query Language (QL)	SQL as Standard QL	No Standard QL (Multiple QLs)
Preferred Distribution	Single Node	Cluster of Nodes
Different Attributes	Not Allowed	Allowed
Join-Operations	Allowed	Avoided
Open-Source	Few Databases	Most Databases
Installation	Simple	Complex
Administration	required	not required
Commercial Support	YES	NO
Comprehensibility	High User-Friendliness	Requires Expertise
Risk of Security Issues	Low	Moderate
Scalability	Scales well vertically	Scales well horizontally

Table 1.: Summary of Differences between SQL and NoSQL Database Systems

Regarding NoSQL databases, there are four different types of core NoSQL databases: key-value stores, document stores, column stores and graph databases. They all have in common that they do not rely on a pre-defined schema and that they scale well in the horizontal direction. The underlying architecture varies from database to database which makes each database type suitable for different areas of applications. Thereby, some databases overlap in terms of suitable employments or functionality. Surely, not all of the four database types will be useful in concern of the data which is gathered by the IOW. Thus in chapter 4, it will be highlighted which databases should be considered with regards to the GODESS data characteristics and which not.

4. Concept

The GODESS project delivers a vast amount of different semi-structured data stored in different files as well as a wide range of functions to choose from. At the same time, the current state of research supplies an enormous number of different database systems including SQL- and NoSQL-based applications to be potentially selected for this approach. Since resources within the scope of this thesis are limited, it will be necessary to scale down from this broad variety of data, functions and systems to a slightly smaller but exemplary transformation model. Therefore, the following chapter explores the methodological design as well as programming languages, sample functionalities, the reference model and data used for illustrating the transformation.

4.1. General Procedure

This section will explore the general procedure. Therefore, Figure 6 shows the methodological design that will be implemented. Given the Matlab-analysis from the IOW, it will firstly be analyzed which functions can be identified inside the scripts. This includes functionalities occurring permanently during the script as well as more specific functionalities with greater significance. Thereby, smaller functionalities with same modalities such as iterative functions will be classified as a standalone group. Simultaneously and based on the current state of research, two NoSQL databases are chosen for the transformation process - Apache's Cassandra and MongoDB. PostgreSQL is selected as a representative for relational database systems. In section 4.2 it will be further explained why the different systems are chosen and how they are characterized. Once the scripts are analyzed and the functionalities are identified, the used database systems and the current SQL-standard will be compared in terms of their capability to realize those Matlab-functionalities.

As a result, each function will be assigned to one out of three categories for each database system. Each category thereby defines a certain degree of technical feasibility. On this occasion, the functionalities will be classified as described in Table 2. Since category one defines functionalities as almost not transformable, the next step regarding those functionalities will be to propose another way to solve this problem. For further exemplary transformation only functions from category two and three will be considered.

1	The Functionality cannot be transformed at all or cannot be realized without making too extensive efforts including highly complex implementations as well as support by additional software.
2	The Functionality cannot be realized by a one-to-one-transformation but can be realized in a different way and with reasonable effort.
3	The Functionality is already realized as a function or operator in the database-language standard and therefore ideally can be realized by a one-to-one-transformation.

Table 2.: Degree of Technical Feasibility

In a next step, the variety of functionalities assigned to categories two and three (*functions 1 to n*) will be scaled down with the result that only two functionalities will be selected (*function x and function y*). Finally, both functions will be transformed to three different database-systems using an exemplary data model illustrating the GODESS data pack. This will involve PostgreSQL as SQL-system as well as Cassandra and MongoDB as NoSQL-systems.

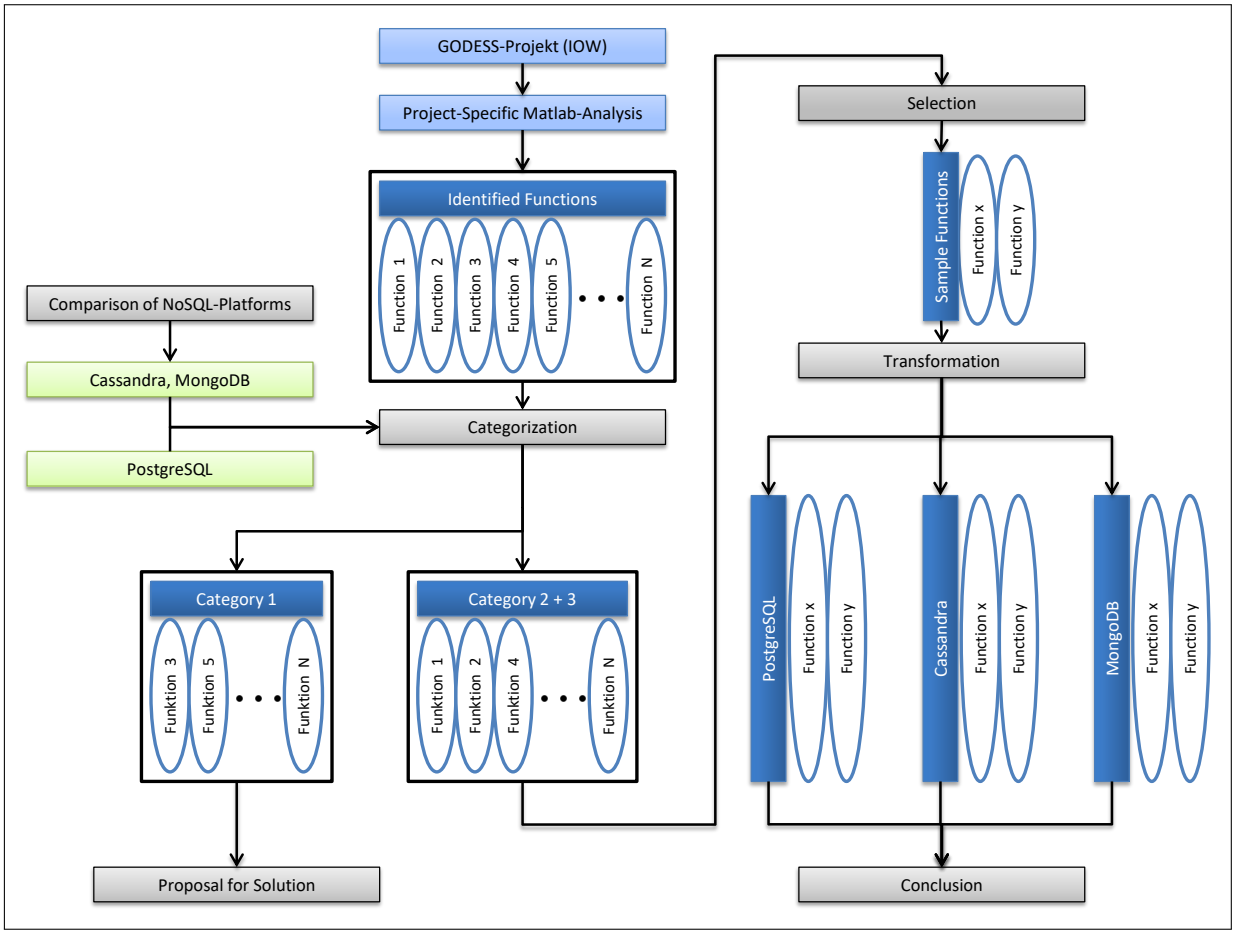

Figure 6.: Methodological Design of Implementation

4.2. Used Programming Languages

Since there is a wide range of database systems to potentially fit the requirements of the GODESS project, it is important to restrict this variety based on the latest state of art. With regards to chapter 3 there are basically two major categories to choose from. This involves SQL-based databases as well as NoSQL database systems. In addition, there are further four different sub-categories of NoSQL databases to be potentially chosen: key-values stores, column stores, document stores and graph databases. In terms of performing an exemplary transformation of a short segment of the Matlab scripts provided by the IOW, three different database systems will be included in the model. This involves a relational DBMS, a column store and a document store. While PostgreSQL is choosen to perform as a relational DBMS, Apache's Cassandra is picked as a representative of column stores and MongoDB as a document store. Since soft NoSQL-systems like object-oriented databases only coincidentally share the traits being

described as NoSQL-characteristics and, therefore, cannot exactly be assigned to the category of NoSQL-databases, they will not be considered. Since key-value stores and graph databases seem not to be as suitable as the remaining NoSQL stores, they will not be considered, as well. In the following, it will be explored why PostgreSQL, Cassandra and MongoDB are choosen and which basic features these databases offer.

PostgreSQL

As it is the goal of this paper to perform a transformation of Matlab functions on both, SQL and NoSQL database systems, it has been necessary to pick a SQL-database covering the majority of the SQL-standard functionalities. PostgreSQL comes along not only with a high level conformity of the SQL-standard, but also with a strong degree in user-friendliness and easy-to-use capabilities. Furthermore, it was used as standard software in the scope of different theses written by students of the University of Rostock that were also dealing with the data gathered by the GODESS in the Godland Basin. [Möl16,Mey16]

PostgreSQL exists since 1995 [VVM12], is an open-source relational DBMS [JYBC15] and can be seen as a heavily used representative of SQL-databases. [VVM12] It is easy to install [JYBC15], fully transactional and ACID compliant [VVM12]. Thereby, it is implemented in a client-server manner. Due to the separation of client and server, the client's library became lighter and it has the advantage, that any changes made in the database engine will not affect the client. [JYBC15] Moreover, the setup of PostgreSQL automatically installs pgAdmin, which is an open-source software for managing PostgreSQL databases offering a userfriendly graphic interface. The database allows declarations and constraints like defined by the SQL-standard. This includes primary keys, alternative keys, not null, referential integrity and attribute and record level validation rules. [FC13]. With PostGIS this database even offers a special extension integrating several geo-functions and geographic objects. [MTSA19]

In PostgreSQL, querying data is mainly realized by using the `SELECT`-statement. [Mom01] Listing 4.1 shows the basic `SELECT`-statement specification. Basically the `SELECT`-statement cannot be executed without the `FROM`. Whereas, the `FROM`-clause defines the target relation to be viewed, the `SELECT`-statement specified which columns will be addressed. Thereby, the user can view single or multiple columns. To view all existing columns of a target relation, the user can specify as column name. Consequently, all rows of the target table will be returned.

```
1 SELECT columnnames
2 FROM tablename;
```

Listing 4.1: SELECT-statement in SQL

PostgreSQL is considered to be the best choice when very flexible query capabilities are needed or read performance is a priority. [VVM12] Based upon real business scenarios and performed on single servers, PostgreSQL also turned out to perform four times faster than MongoDB in terms of response time. [MTSA19] In conclusion, PostgreSQL is a perfect fit for being a representative of relational and SQL-based databases.

Choice of NoSQL Systems

Concerning NoSQL-database-systems, two different types of stores seem to meet the requirements of the GODESS data. Based on the latest state of art and data store characteristics, column stores and document stores both seem to be appropriate fits. Since Matlab is superficially used as analytic tool and column stores are considered highly reliable in applications dealing with data analytics, a column

oriented database will be a proper system to rely on. Therefore, Apache's Cassandra is used, because of its SQL-similarity in syntax and its popularity within the NoSQL community. As the GODESS data is stored in TXT-files or in XML-format and in a semi-structured manner, document stores seem promising as well. Consequently, MongoDB is used as a representative of document oriented databases. According to the research community, it is not only the most popular document database [KR13], but also cemented its place as the most popular NoSQL database in general. [MH13] The popularity along with a huge repertoire of functionality lead to the choice of MongoDB as a second NoSQL-system. In contrast, key-value-stores and graph databases were not considered. This is because most column stores easily include the characteristics as well as extend the capabilities of simple key-value-stores and, additionally, graph databases perform better in analyzing relationships among the data, which is not essential in this case.

Cassandra

Cassandra was developed by the Apache Software Foundation and exists since 2008. [VVM12] It takes its features from both Google's BigTable and Amazon's Dynamo. [KR13] This makes it a hybrid database bringing together Dynamo's fully distributed large design and BigTable's column family based data model. [TB11] Thus, concepts of both key-value-stores and columns stores are involved. It is written in Java and similar to usual relational models, but data that need to be stored can be structured as well as semi-structured or unstructured due to its dynamic schema. [NPP13] The query language is Cassandra Query Language (CQL) and offers a SQL-similar syntax. Cassandra relies on BASE properties, which makes it a AP type system regarding the CAP-theorem. [VFKV16]

The data model in Cassandra consists of keyspaces, column families, keys and columns. Before tables are created, the user needs to specify a specific namespace called keyspace along with a specific partition strategy. Tables are then created within the defined keyspace. Cassandra partitions data across the cluster by using consistent hashing. [LM10] Thereby it is important to choose a proper partition strategy. Currently, Cassandra provides two separate partitioners. The random partitioner distributes the key-value pairs randomly over the network. In contrast, the order-preserving partitioner distributes the key-value pairs in a natural way so that similar keys are not far away from each other. [Apa14]. Generally, each key corresponds to a value which represents a highly structured object. [LM10] Thus, the rows are represented in a key-value-pair manner. Thereby, the corresponding object can be distributed over multiple columns. These columns then are grouped together into sets called column families. Additionally, these column families can be grouped together once again into super column families. [Apa14] The column families can be referred to tables in a relational DBMS. Hence, a table in Cassandra is a distributed multi dimensional column-oriented map indexed by a key. [LM10]

Like SQL, CQL supports range queries. Additionally, it scales horizontally, supports linear expansion and provides replication and sharding. [HHLD11] It offers multi-master replication that allows more than one master at the same time and multiversion concurrency control. [AB13] Thereby, a write operation will be replicated to other nodes while a read request will be routed to a specific target node. [HHLD11] Next to this, Cassandra as a DBaaS provider is programmed to handle hardware failures. [NPP13]

At first sight, querying data in Cassandra seems to function just like in SQL. The SQL-like syntax implies, that CQL can be used evenly although it is not the case. As long as there are no further restrictions or conditions on the `SELECT`-statement from Listing 4.1, there is no reason to differentiate. But when it comes to more complex queries (like it will be explored in section 4.3), the user needs to face some differences. Particularly when it comes to specifying the primary key, users need to be careful in order to guarantee efficient querying performance, since there are two different types of keys than can

potentially be specified - partition keys and clustering keys. Whereas the partition key is responsible for distributing the data among the nodes of the cluster, the clustering key is responsible for sorting the data within a partition.[LM10]

According to Li and Manoharan[LM13], it performs slower than MongoDB in terms of read, write and delete. Furthermore, they found that Cassandra is slower than the relational database SQL Express in terms of read and delete, but outperforms it in write operations. In contrast, Cassandra is faster than MongoDB when it comes to update operations due to the performed replication technique and is considered to increase its performance with the increase of the data set size.[AB13] Among other characteristics, this makes Cassandra the best choice for large critical sensor applications[VVM12] and an optimal fit for storing and interacting with large amounts of data.[AB13]. Subsequently, Apache's hybrid store is applied in social networking, on websites, in banking and finance, for real-time data analytics and by online retailers.[NPP13]

MongoDB

MongoDB is a multi-platform database developed in 2007 as an open-source project by 10gen having its first public release in 2009.[AB13] The word "mongo" originates from the term "humongous"[KR13] and refers to the vast amounts of data that can be processed with MongoDB. It is written in C++ and can be classified as a schema-free document database.[SK11] The main goal of MongoDB was to close the gap between the fast and horizontally scalable key-value-stores and feature rich relational DBMS, which is why this data store can be seen as a non-relational database featuring the richest and most common operations of relational DBMS.[HHLD11] The query language is called Mongo Query Language[NPP13] and allows to include regular expressions as well as JavaScript functions.[VVM12] Like Cassandra, MongoDB relies on BASE properties, but is rather a CP type system than an AP type database.[AB13]

In MongoDB, the data is stored in BSON documents.[AB13]. The documents contain an ordered list of elements consisting of field name, type and value.[NPP13] Thereby, the data is stored with self-contained records and no intrinsic relationships.[OBLB15] Each document is identified by a unique ID and documents can be grouped together into so called collections.[AB13] Collections can be referred to tables in a relational DBMS or to column families inside keyspace in Cassandra. In contrast, each collection can be composed of documents having a completely different structure.[FC13] Furthermore, it is possible to nest documents inside each other and to reference documents.[MTSA19] Generally, collections are created on the fly whereas documents are inserted or updated using functions provided by MongoDB's API.[FC13]

Unlike CQL, mongo querying language has no SQL-similar structure. Since, MongoDB is different in both data model and querying language, the basic querying command looks different from the SELECT-statement. Listing 4.2 illustrates how data in MongoDB can be addressed using the $find()$-operator. Firstly, a database (*db*) needs to be created. Once it is created, a collection is created on the fly by the user. Finally, this collection will be queried by using the name of the collection appended by the $find()$-statement. The statement can be seen as equivalent to SQL's `SELECT`-statement and offers a wide range of possibilities to further specify the query including JavaScript functions and a powerful aggregation framework.[Tre14]

```
1 db.<collection>.find()
```

Listing 4.2: $find()$-statement in MongoDB

In addition to horizontal scalability, it comes along with replication, a high performance aggregation framework, strong persistence [SK11] and a good community support. [OBLB15] On top of that, MongoDB provides GridFS as distributed file system. [KR13] The replication is performed using a master-slave replication mechanism. [AB13] Establishing multiple slaves but only one master, the master is allowed to write and read the files while the slaves serve as backup. When the master commits, the slave with the most recent data becomes the new master. Next to this, MongoDB allows dynamic queries [OBLB15] as well as multi-attribute look-ups on records may containing different kinds of key-value pairs. [MTSA19]. Furthermore, it uses locks to ensure consistency and prevent multiple clients to read and update data at the same time. [AB13]

Latest research implies that MongoDB is a high performer among NoSQL database systems. It is ten times faster in terms of access speed than the relational and SQL-based DBMS MySQL. [HHLD11] Furthermore, MongoDB is faster than SQL Express in concerns of the creation of database buckets and better in read, write and delete performance compared to SQL Express and Apache's Cassandra. [LM13] In contrast, it decreases in performance with the increase of the data set size. [AB13] Additionally, it turned out that MongoDB performes faster in operating insert, select, update and delete than PostgreSQL. [JYBC15] Apart from that, MongoDB is considered to be the best choice for small or medium-sized non-critical sensor applications especially when write performance is a priority. [VVM12] Based on its characteristics, MongoDB is suitable for content management systems relying on dynamic queries [AB13], mobile applications, gaming and archiving [KR13] or real-time data analytics. [NPP13]

Summary

Finally, three different database-systems are used. Thereby, one SQL-based system and two NoSQL-databases out of different NoSQL categories are choosen. Despite the difference in schema and data model, there are some parallels that could be observed between all databases. Therefore, Table 3 compares each part of the relational data model its analogue in the data models of Cassandra and MongoDB. Whereas the complete data set is called database in both PostgreSQL and MongoDB, Cassandra uses keyspaces. Furthermore a relation can be referred to a column family inside a Cassandra keypace and to a collection of documents inside a MongoDB database. In PostgreSQL the data is identified by a primary key while in Cassandra a set of various partition and clustering keys can be specified and MongoDB provides document identifier. Thereby, the actual data is stored either as tuples, as key-value-pairs or as documents.

PostgreSQL	**Cassandra**	**MongoDB**
Database	Keyspace	Database
Relation	Column Family	Collection
Primary Key	Partition/Clustering Keys	Document-ID
Tuple	Key-Value-Pair	Document

Table 3.: PostgreSQL, Cassandra and MongoDB: Data Models and Designation

All together, PostgreSQL, Cassandra and MongoDB form the basis of database systems for this approach. PostgreSQL has been chosen based on familiarity and Cassandra and MongoDB were chosen based on the latest state of art and popularity. The programming languages will transform two different sample functionalities identified in the original Matlab script. Therefore, section 4.3 will categorize the identified functionalities based on the defined degree of technical feasibility. Furthermore, it will be explained how each identified functionality can be basically realized and how expensive the realization is.

4.3. Categorization and Comparison of Functionalities

As already mentioned, the Matlab-analysis performed by the research group "chemical in-situ sensors" at the IOW is based on three Matlab scripts consisting of an enormous volume of mathematical operations and functions. Thereby, the main script consists of twelve steps including the loading and processing of the data files, comparing different CTD casts, plotting graphs and timelines and the deconvolution of slow sensor data. In two additional scripts, practical and absolute salinity are calculated. Since, most operations particularly in the main script depend on prior steps involving complex calculations and algorithms, it is not possible to describe every single component of the script in detail nor to analyze every operation in terms of transformability. Therefore, the Matlab scripts were screened for functionalities occurring frequently or with special significance for the analysis results. With this in mind and like already described at the end of section 2.3, there is a bunch of functionalities that need to be considered.

Table 4 shows the lists of functionalities detected in the Matlab scripts and compares them based on the defined degree of feasibility. Primarily, ten different elementary functionalities can be distinguished. This includes permanent operations like complex case differentiation, iteration, sort and merge algorithms or element-wise and matrix multiplication, but also specific and highly significant functions such as range queries, polynomial evaluation, variance analysis, multiple linear regression and plotting functions for visualizing data as diagrams.

Based on which degree of feasibility a specific functionality refers to, the cell is colored green, yellow or orange. As already mentioned in Table 2, degree three is the worst degree and is assigned to a functionality that cannot be transformed at all or cannot be realized without making too expensive efforts such as complex implementation or support by additional software. On the contrary, degree two defines that functionalities assigned to this degree cannot be realized by a one-to-one-transformation but in a different way and with reasonable effort. Finally, degree 1 is the best degree and is assigned to a functionality that is already realized as a prefabricated function or operator in the database-language standard and therefore ideally can be realized in a one-to-one-transformation.

At this point, it is important to notice, that there is no functionality that could not be assigned to any of the three defined degrees. Thus, every single functionality can be transformed in one way or another, but each transformation requires a specific level of effort to be put in. Therefore, the following section will picture how those ten functionalities can potentially be realized in the different used query languages and what difficulties need to be faced to perform the realization.

FUNCTIONALITY	**SQL-Standard**	**PostgreSQL**	**Cassandra**	**MongoDB**
Complex Case Differentiation	2	2	2	2
Iteration	2	2	2	2
Sorting	3	3	2	3
Merging	3	3	2	3
Matrix Multiplication	2	2	1	1
Range Querying	3	3	3	3
Polynomial Evaluation	2	2	2	2
Variance Analysis	3	3	2	2
Multiple Linear Regression	1	1	1	1
Plotting Diagrams	1	1	1	1

Table 4.: Identified Functions: Comparison across Database-Systems based on Degree of Feasibility

Complex Case Differentiation is one of the functionalities operated permanently during the Matlab-analysis. The SQL-standard delivers two statements (`IF` and `CASE`) that can be used to differentiate between two cases. Furthermore, optional `ELSE IF`-clauses are allowed. The `CASE`-statement then can be used to extend the IF-statement. According to the IBM documentation, both statements are generally used to conditionally enter into some logic based on simply the status of condition. [IBM15] Another opportunity to realize any more complex case differentiation is nesting `SELECT`-statements. Because the statements are included in PostgreSQL, these methods can be performed on PostgreSQL servers as well. Hence, simple case differentiation can be easily produced by using `IF` or `CASE`. Since the case differentiation in the Matlab analysis is rather complex, this functionality needs to be assigned to category two.

In MongoDB or Apache's Cassandra there are no statements being equivalent to `CASE` or `IF`. But since MongoDB supports JavaScript functions, complex case differentiation can be realized this way. The same applies to Cassandra. SQL-similar query language CQL does not provide statements like `CASE` or `IF` and, therefore, complex case differentiation can only be realized by nesting `SELECT`-statements or by creating a UDF in Java, JavaScript or another installed programming language that is choosen. In conclusion, there is no way to implement complex case differentiation like being presented by the Matlab scripts in a one-to-one-manner.

Iteration is similarly difficult to achieve. In PostgreSQL, iteration can be performed in a one-to-one-transformation, if no mathematical operations are performed within the iteration. Otherwise, there is a danger of producing endless loops over the database. Both the SQL-standard and PostgreSQL provide four different statements for iteration. This includes `FOR`, `LOOP`, `REPEAT` and `WHILE`. The `FOR`-statement is considered distinct from the other statements, as it is used to only iterate over rows of a defined result set whereas the remaining statements can be used to iterate over a series of SQL-statements until for each a specific condition is satisfied. Furthermore, `SELECT` can be used recursively within another `SELECT`-statement. [IBM15] `LOOP` represents the basic version of an iteration and is a special type of iterative statement, because it does not include a terminating condition clause. Apart from that, the `WHILE` statement defines a set of statements to be executed until a condition that is evaluated at the beginning of the `WHILE` loop is false. [IBM15] In contrast, the `REPEAT`-statement defines a set of statements to be executed until a condition that is evaluated at the end of the `REPEAT` loop is true. [IBM15] Generally, there can be used multiple loops at the same time. Thereby, extern variables can be declared within the loop by using the `DECLARE`-statement.

In MongoDB, iteration can either be realized by JavaScript functions or by the $forEach()$-Operator within a query. Thereby, the operator is simply added to the $find()$-statement and, subsequently, iterates over a specific collection of documents. Hence, there is a one-to-one-statement for realizing iteration within a collection, but this will not be sufficient to cover the complexity of iterations used in the Matlab files. Regarding Cassandra, there is no equivalent to the variety of loop-statements provided in relational database system relying on SQL. Thus, iteration can only be realized by implementing UDFs in a foreign programming language or by complex nesting of `SELECT`-statements.

Sorting illustrates the first functionality that reaches different degrees of feasibility over the used systems. In terms of SQL-based applications and MongoDB, sorting can be realized in a one-to-one-transformation. In contrast, it is just really not possible to make Cassandra sort the data by an arbitrary column, since Cassandra requires a query-based modeling approach, which means that the can only order by the defined clustering key within a specific partition.

The SQL-standard and PostgreSQL come along with a sorting `ORDER BY`-statement that can be added to the `SELECT` environment. It simply orders the relation in an ascending or descending order by single

or multiple columns. Likewise, MongoDB provides the $*sort*-command that can simply be added to the $find()$-statement and can be seen as an equivalent to SQL's `ORDER BY`-statement. Although Apache's Cassandra provides `ORDER BY` as well, this statement is no equivalent to the SQL original. In Cassandra, the `ORDER BY`-clause can only be used to reverse the defined sort direction of a clustering key. If the user needs to sort the data based on a specific column, this functionality can only be realized by specifying a clustering order. Cassandra then fully automatically orders each partition based on the defined clustering key.

Merging of different data sets is a functionality that is well developed across database applications. Database systems perform joins or other techniques to achieve assembling various pieces of information together. As it is the case in SQL, the Standard and PostgreSQL both deliver a rich set of `JOIN`-statements. This includes `INNER JOIN`, `FULL JOIN`, `LEFT JOIN` or `RIGHT JOIN`. Thereby, `INNER JOIN` selects all tuples that are included in both tables while `FULL JOIN` selects all tuples that are included either in the first table or in the second table. Additionally, `LEFT` or `RIGHT JOIN` select dangling tuples which are those tuples in the referencing relation that do not have counterparts in the referenced relation. Another statement provided by SQL presents the `UNION`-statement. Whereas `JOIN` combines the data into new columns, `UNION` rearranges the data in new rows and, thus, combines the result sets of two queries by column position rather than column name if the column data type in both queries are matching. [IBM15]

In contrast to SQL-based database systems, NoSQL databases try to avoid join operations and, therefore, are not designed to perform such operations efficiently. As it is the case for Cassandra, join operations as known in SQL are simply not possible. Similar results can only be achieved by developing an appropriate data model considering every data combination needed for efficient future querying or by implementing complex algorithms via programming interfaces. Nevertheless, some NoSQL databases already provide similar techniques like joins. MongoDB, for instance, comes with the $*mergeObjects*-command which enables the user to combine an arbitrary number of documents into a single document. Hence, documents can be bound together similarly to relations in a SQL-based DBMS.

Matrix Multiplication remains an issue when it comes to performing such operations efficiently on a database server. Since most database systems are not providing a matrix data type, element-wise vector or matrix multiplication cannot be realized in a one-to-one-manner. Marten and Heuer [MH17] have shown that matrix multiplication can be realized with reasonable effort in SQL by using a combination of aggregate operators and join functions. Since NoSQL databases store their data in a non-rigid schema, this method cannot be transferred effortlessly to Cassandra or MongoDB. As a result, the schema free model of Cassandra and MongoDB impedes that matrix operations can be implemented without relying on other programming language interfaces or complex algorithms.

Range Querying obviously can be easily achieved across all different kinds of database systems. Since most databases are purposely designed for querying data efficiently in all kinds of different manners, SQL, CQL and Mongo Querying Language support range queries. Like already depicted in section 4.2, SQL queries data using the `SELECT`-statement. Within this statement, the `WHERE`-clause can be added to specify a querying condition that refers to single or multiple relation variables which can be expressed as a range. Equivalently to the `WHERE`-statement in PostgreSQL, MongoDB provides the $*where*-operator which can be appended to the $find()$-statement for creating views satisfying a specific querying condition. Inside the queried collection, each document that satisfies the condition is returned. CQL also serves the `WHERE`-clause but due to Cassandra's dynamic schema, there are some restrictions on it. Consequently, the `WHERE`-clause can only be used on the defined keys. Hence, the user once again needs to know precisely which column need to be queried before the schema is developed.

Polynomial Evaluation constitutes a functionality that can be realized with adequate expense in every database language used. Depending on the polynomial's degree, this expense can be on a different level. Thus, the expense increases with the degree. In the SQL-standard and PostgreSQL, polynomial evaluation can be realized by nesting `SELECT`-statements. Polynomials can be even calculated within a single `SELECT`-statement as long as the polynomial can be implemented as a scalar expression. Additionally, UDFs can be implemented which composes the smarter method for evaluating high-degree polynomials. Some SQL-based databases do even provide prefabricated functions. Microsoft Server SQL, for instance, comes with a `POLYVAL`-statement that calculates polynomials automatically by a given input.

Regarding Cassandra's CQL, UDFs remain the only possibility to evaluate polynomials, since there is no possibility of creating scalar expressions inside the `SELECT`-statement. In contrast to CQL, MongoDB provides more flexibility. Although MongoDB provides the possibility of saving UDFs implemented in JavaScript in order to later apply them on specific documents inside a collection, polynomial evaluation can be executed even within a query by defining an intern UDF inside the $find()$-statement as long as the polynomial can be evaluated as a scalar expression. Thereby, the user can create a UDF by specifing the following prototype form the way as desired: $db.collection.find().forEach(< function >)$.

Variance Analysis can be transformed easily as well. In the SQL-standard as well as in PostgreSQL, there are two prefabricated statements that can be used. This involves `VAR_POP` for calculating population variances and `VAR_SAMP` for estimating sample variances. In terms of Cassandra and MongoDB, there are no prefabricated statements to use. Instead, variance analysis can be implemented similarly like described for the case of polynomial evaluation. Thus, analyzing variances of different variables and parameters is a lot more easier in SQL, but can be realized with low effort in each database system.

In contrast to the other functionalities, **Multiple Linear Regression** and **Plotting Diagrams** remain difficult to achieve on a database level. Due to the complexity of multiple regression analysis, this major functionality can only be achieved by implementing complex programs inside the databases' aggregation frameworks, creating highly complex UDFs or by relying on additional software and programming language interfaces. The same applies for plotting diagrams of single or multiple variables. Although data and the related traffic can often be monitored, database systems are not purposely designed to create highly specific graphs calling for an enriched color palette and a broad range of design and layout options. Conclusively, multiple linear regression and plotting diagrams in a standard required for the analysis performed at the IOW cannot be transformed without producing unnecessarily high effort.

Comparing PostgreSQL, Cassandra and MongoDB based on the defined degree of technical feasibility shows that the majority of functionalities can be covered by SQL-based database systems. Although NoSQL systems like Cassandra and MongoDB already provide a wide range of functionalities enabling the user to transform Matlab analysis into database-supported evaluations, the SQL-standard and PostgreSQL clearly seem to outperform both NoSQL database types in terms of feasibility, since there are more functionalities that could be assigned to a feasibility degree of two or even three. Neither a combination of column and key-value store as it is the case with Cassandra nor MongoDB as a representative for document stores seem to achieve the same level of feasibility like the relational "dinosaur-system". In regard to highly complex statistical methods or graphical functionalities as seen with multiple linear regression and plotting diagrams, each used database system fell short of expectations. In conclusion, there is no database system fully covering the entirety of required functionalities included in the IOW Matlab scripts, but the potential of transformation is particularly high throughout all used query languages.

This section provided an overview about the variety of functionalities and explored how they can be realized across the different database systems and how reasonable this realization is. Since, this thesis

focuses on implementing only a part of the functionalities involved in the Matlab anaylsis, two functionalities out of the entire set will be chosen. Therefore, section 4.4 will explore which functionalities are chosen for which reasons. To simplify the demonstration, an exemplary data model will be developed. Thus, section 4.4 also depicts how this data model is characterized and which data will be included.

4.4. Sample Functionalities and Exemplary Data Model

Like section 4.3 implies, there is a wide range of functionalities that can potentially be transformed from Matlab-analysis programs into database-supported evaluation with reasonable effort. On top of that, we have seen in chapter 2 that the data-files produced by the GODESS powerlogger contain vast amounts of data that are processed in Matlab script consisting of highly interdependent steps. Although some Matlab sections can run autonomously, they mostly depend on intermediate results. Furthermore, section 4.3 has shown that not every functionality can be transformed on a database level without undertaking expensive effort. Therefore, it will not be the goal to perform a transformation of the entire Matlab-analysis, but to give an exemplary illustration of how selected Matlab programs could be transformed to PostgreSQL, Cassandra and MongoDB. As a consequence, only two functionalities will be implemented. Thereby, an exemplary data model is used for demonstrating the transformation of the chosen functionalities.

Chosen Functionalities

Based on the comparison in section 4.3, sorting and polynomial evaluation are chosen as sample functionalities for several reasons. Firstly, both functionalities together are covering the entire set of defined degrees of technical feasibility that can be realized with reasonable effort. This is because sorting could be assigned to degree one or two and polynomial evaluation to degree two. Secondly, there is a difference between sorting and polynomial evaluation in the number of distinct degrees of feasibility that can be achieved. While sorting was assigned to two different degrees of technical feasibility, polynomial evaluation achieves the same degree of technical feasibility for all uses database systems. Consequently, implementing the sorting functionality provides an excellent example of demonstrating differences between the uses systems whereas the choice for implementing polynomial evaluation provides an adequate basis for comparing the used database systems on the same level. All in all, the choice of the functionalities ensures that the data model provides strongly representative character.

FUNCTIONALITY	**SQL-Standard**	**PostgreSQL**	**Cassandra**	**MongoDB**
Sorting	3	3	2	3
Polynomial Evaluation	2	2	2	2

Table 5.: Chosen Functions: Comparison across Database-Systems based on Degree of Feasibility

Included Data

Table 6 shows the exemplary data model that has been composed for the approach of transformation. It consists of eight variables. Thereby, four variables derive from the CTD-file whereas the other four variables derive from the metadata structures provided by the XML-file. To simplify the data model, not all variables included in the CTD-file were chosen. Subsequently, the data model focuses on data being necessary for estimating conductivity. Hence, polynomial evaluation will be performed only for calculating the actual conductivity based on the measured conductivity values provided by the CTD-probe. Generally, the data model contains 18 individual records.

profile_id	deployment_id	ts	conductivity_value	concoeff_1	concoeff_2	concoeff_3	concoeff_4
201	8	2009-08-22T14:07:22.643	11504	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
201	8	2009-08-22T14:07:22.869	11502	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
201	8	2009-08-22T14:07:23.125	11502	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
201	8	2009-08-22T14:07:23.407	11500	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
201	8	2009-08-22T14:07:23.665	11499	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
201	8	2009-08-22T14:07:23.924	11498	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016
434	12	2014-04-02T00:00:00.210	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-02T00:00:00.487	14509	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-02T00:00:00.992	14508	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-02T00:00:01.249	14509	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
435	12	2014-04-02T00:00:01.553	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
435	12	2014-04-02T00:00:01.791	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:58.623	14514	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:58.834	14514	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:59.031	14513	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:59.277	14512	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:59.606	14512	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016
434	12	2014-04-01T23:59:59.892	14511	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016

Table 6.: Exemplary Data Model

The first two columns (**profile_id** and **deployment_id**) demonstrate the profile number and the number of deployment. Thereby, the profile number is defined as a counting variable numbering all profiles consecutively over all deployments and the number of deployments counts all deployments chronologically. The deployment ID refers to the deployment number in the XML-file while the profile ID basically replaces the $PSDA < number >$-String which constitutes the beginning of each row of the CTD-file's payload (see Listing 2.1) by an integer only displaying the $< number >$-part of it.

The third column (**ts**) presents the timestamp of every single measurement and, therefore, combines multiple CTD-file variables rolled into one. This involves DATE (YY-MM-DD), TIME (HH:MM:SS) and FRAC (MS) which are stored together in a single timestamp (YY-MM-DD HH:MM:SS.MS). Consequently, the timestamp shows the year (YY), the month (first MM), the day (DD) as well as the hour (HH), the minutes (second MM), the seconds (SS) and the milliseconds (MS) in which each measurement was performed.

The remaining columns contain converted 16-bit-integer values for conductivity (**conductivity_value**) as well as the four calibration values generated for the conductivity sensor and provided by the XML-file (**concoeff_1-4**). As already mentioned in section 2.2, both are inevitable for polynomial evaluation which means that the 16-bit-integer values represent a polynomial coefficient which can be used to compute the actual conductivity values based on the given calibration coefficients. For further analysis, the integer values will be converted and stored in float-format.

As Table 3 indicates, the exemplary data model consists of three different parts or day segments. While the first part belongs to one deployment (deployment 8), the remaining two parts belong to another deployment (deployment 12). This ensures that two different sets of calibration values are used for the polynomial evaluation. Moreover, the data is not sorted by timestamp in any case and profile number 434 is split over two different days exceeding the date the measurement was started. By including the case of date exceedance, the model ensures that the data requires to be resorted.

All in all, it need to be mentioned that the data model provides a strongly exemplary character and primarily was designed with the intention to illustrate basic transformation possibilities of the used database systems. Due to the limited time and resources as well as high complexity of both Matlab code and data structures, a fully developed model cannot be applied. Therefore, two functionalities including sorting and polynomial evaluation for conductivity have been picked and the fictional data model was composed in orientation of parts of two different data files including the TXT-file generated for the CTD multiparameter probe and the XML-file manually generated by the IOW researchers.

In summary, two different functionalities will be transformed based on an exemplary data model across three different database systems. This involves PostgreSQL, Apache's Cassandra and MongoDB. chapter 5 will illustrate how sorting and polynomial evaluation are transformed from Matlab to database-supported evaluations on the used systems. Thereby, a Matlab reference model will be provided to ensure better comparability of the output produced by each system.

5. Implementation

Based on the used programming languages and the exemplary data model, the following chapter will explore how the sample functions involving sorting and polynomial evaluation are realized. First of all, a reference model will be implemented. The reference model will illustrate how sorting and polynomial evaluation can be realized in Matlab. Based on the reference model, the same functionalities will be implemented in PostgreSQL, Cassandra and MongoDB. Before the implementations are pictured, it will be stated how the data model is constructed on each of the different database systems. Thereby, execution time is measured for both sorting and polynomial evaluation independently on all used systems. This includes Matlab as well as all database-supported evaluations.

5.1. Reference Model

Before sorting and polynomial evaluation will be performed on a database-supported level, a reference model will be implemented in Matlab. Since Matlab's octave is a matrix and vector based programming language, the data model is implemented as a vector based approach. Although Matlab also offers the ability to work in database-similar table environments, this environment will not be considered, since the original Matlab scripts written by the IOW research group do work on a matrix and vector basis, as well. As derived from the original Matlab scripts, the data is stored in column vectors. Listing 5.1 exemplary shows how those column vectors are realized for `profile_id`, `deployment_id` and `concoeff_4`. Thereby, the first part of the data model is designated as `"_old"` whereas the other two parts are designated as `"_new_b"` and `"_new_a"`.

```
1  profile_id_old = [201;201;201;201;201;201]
2  profile_id_new_b = [434;434;434;434;435;435]
3  profile_id_new_a = [434;434;434;434;434;434]
4
5  deployment_id_old = [8;8;8;8;8;8]
6  deployment_id_new = [12;12;12;12;12;12]
7
8  concoeff_4_old = [0.000000000000000120393922;0.000000000000000120393922;
9                    0.000000000000000120393922;0.000000000000000120393922;
10                   0.000000000000000120393922;0.000000000000000120393922]
11 concoeff_4_new = [0.000000000000000110249781;0.000000000000000110249781;
12                   0.000000000000000110249781;0.000000000000000110249781;
13                   0.000000000000000110249781;0.000000000000000110249781]
```

Listing 5.1: Data as Column Vectors in Matlab

After all vectors are constructed, related vectors are further bound together to create a hyper-vector for each variable. Listing 5.2 demonstrates how this is implemented in Matlab by using the same syntax as before. As a consequence, eight variable vectors are created. Hence, there are vectors for `profile_id`, `deployment_id`, `ts`, `conductivity_value` and all of the four conductivity coefficients. So far, they are still unsorted. But now the vector model is ready to be processed. In the next steps, it will be shown

how those vectors can be sorted on an index level and how polynomial evaluation can be performed in order to demonstrate the computation of the actual conductivity values.

```
profile_id = [profile_id_old;profile_id_new_b;profile_id_new_a]
deployment_id = [deployment_id_old;deployment_id_new;deployment_id_new]
ts = [ts_old; ts_new_b; ts_new_a]
conductivity_value = [cond_old; cond_new_b; cond_new_a]
concoeff_1 = [concoeff_1_old;concoeff_1_new;concoeff_1_new]
concoeff_2 = [concoeff_2_old;concoeff_2_new;concoeff_2_new]
concoeff_3 = [concoeff_3_old;concoeff_3_new;concoeff_3_new]
concoeff_4 = [concoeff_4_old;concoeff_4_new;concoeff_4_new]
```

Listing 5.2: Binding Vector Parts to Variable Vectors

Once the data model is implemented, the vectors can be sorted. Therefore, the user needs to know by which variable should be sorted. In case of the used data model, the `ts`-variable is the only variable with unique values. Hence, the data should be sorted by timestamp and the order of the `ts`-vector will set the order for the remaining vector variables. As shown by Listing 5.3, `ts` is sorted and the index of the new sorted vector `S` is saved as `Idx`. Subsequently, each of the remaining vector variables is sorted based on this index. Table 7 shows the output for each vector in one table. It is clearly recognizable that the data model now is sorted by timestamp.

```
tic;
[S,Idx] = sort(ts)

ts = S
profile_id = profile_id(Idx)
deploymend_id = deployment_id(Idx)
conductivity_value = conductivity_value(Idx)
concoeff_1 = concoeff_1(Idx)
concoeff_2 = concoeff_2(Idx)
concoeff_3 = concoeff_3(Idx)
concoeff_4 = concoeff_4(Idx)
toc;
```

Listing 5.3: Sorting Vectors on an Index Level in Matlab

After the data was brought to order, conductivity can be computed by performing polynomial evaluation on the sorted data model. Listing 5.4 illustrates how the conductivity is calculated based on the vector data of `conductivity_value` and the four different calibration values. The computation is performed by implementing a scalar expression. The last column of Table 7 displays the vector representing the output of the polynomial evaluation. It can be noted that the computation has produced realistic output which is ordered chronologically. The vector containing the results produced by the reference will form the basis of comparison for the results produced by the following database-supported evaluations.

```
tic;
conductivity = concoeff_1 + conductivity_value .* (concoeff_2 + conductivity_value.*(
    concoeff_3+conductivity_value.*concoeff_4))
toc;
```

Listing 5.4: Polynomial Evaluation in Matlab

Vector 1	*Vector 2*	*Vector 3*	*Vector 4*	*Vector 5*	*Vector 6*	*Vector 7*	*Vector 8*	*Result*
profile_id	**deployment_id**	**ts**	**conductivity_value**	**concoeff_1**	**concoeff_2**	**concoeff_3**	**concoeff_4**	**conductivity**
201	8	2009-08-22T14:07:22.643	11504	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3177
201	8	2009-08-22T14:07:22.869	11502	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3152
201	8	2009-08-22T14:07:23.125	11502	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3152
201	8	2009-08-22T14:07:23.407	11500	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3128
201	8	2009-08-22T14:07:23.665	11499	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3115
201	8	2009-08-22T14:07:23.924	11498	9.22283727e-002	1.23643726e-003	9.99383726e-012	1.20393922e-016	14.3103
434	12	2014-04-01T23:59:58.623	14514	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8138
434	12	2014-04-01T23:59:58.834	14514	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8138
434	12	2014-04-01T23:59:59.031	14513	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8128
434	12	2014-04-01T23:59:59.277	14512	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8117
434	12	2014-04-01T23:59:59.606	14512	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8117
434	12	2014-04-01T23:59:59.892	14511	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8107
434	12	2014-04-02T00:00:00.210	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8097
434	12	2014-04-02T00:00:00.487	14509	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8087
434	12	2014-04-02T00:00:00.992	14508	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8077
434	12	2014-04-02T00:00:01.249	14509	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8087
435	12	2014-04-02T00:00:01.553	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8097
435	12	2014-04-02T00:00:01.791	14510	8.86499939e-002	1.01438668e-003	9.35514578e-012	1.10249781e-016	14.8097

Table 7.: Results of the Reference Model

5.2. Prerequisite: Constructing the Databases

In contrast to Matlab, database systems do not rely on matrix structures. They rely on totally different data models that can be of different type. Depending on the database type, the database system comes with different schema characteristics that were discussed in chapter 3 and section 4.2. Therefore, each database system has its individual way to store, manage and process data. In this comparison, one relational database and two non-relational databases are included. Since relational databases use strict schemata to store their data and both non-relational databases provide almost schema-less data modeling, there will be some differences in creating the data model. Although all databases use query language, the query languages are highly individual and each provides another syntax. Therefore, the following section will explore how the exemplary data model is created as a relational model in PostgreSQL, as a column family in Cassandra and as a collection of documents in MongoDB.

PostgreSQL

As already stated, PostgreSQL is a relational DBMS using SQL and, therefore, relies on a rigid data model. Because of the rigidity, tables that are created have strict characteristics and data to be inserted has to follow the defined constraints. Listing 5.5 shows how the exemplary data model is realized in SQL by using the `CREATE TABLE`-statement to create a new relation. Since each relation in SQL requires a primary key and timestamp represents the only variable with unique values, **ts** is defined as primary key. Listing 5.6 displays an example of how the data is subsequently inserted by using the `INSERT INTO`-statement. Thereby, the columns of the target table to be filled with data are addressed (lines 2 and 3). Secondly, the data following strict constraints is inserted by using the `VALUES`-clause. After all entries of the exemplary data model are inserted, the relational data model is finally realized.

```
1  CREATE TABLE public.sampledata
2  (
3      profile_id integer,
4      deployment_id integer,
5      ts timestamp(10) without time zone,
6      conductivity_value real,
7      concoeff_1 real,
8      concoeff_2 real,
9      concoeff_3 real,
10     concoeff_4 real,
11     PRIMARY KEY (ts)
12 );
```

Listing 5.5: Relational Model in PostgreSQL

```
1  INSERT INTO public.sampledata(
2          profile_id, deployment_id, ts, conductivity_value, concoeff_1, concoeff_2,
               concoeff_3, concoeff_4)
3          VALUES
4          (201, 8, TIMESTAMP '2009-08-22 14:07:22.643', 11504, 0.0922283727, 0.00123643726,
                0.00000000000999383726, 0.000000000000000120393922),
5          (434, 12, TIMESTAMP '2014-04-01 23:59:59.892', 14511, 0.0886499939,
               0.00101438668, 0.00000000000935514578, 0.000000000000000110249781);
```

Listing 5.6: Creating Tuples in PostgreSQL

Cassandra

In contrast to PostgreSQL, Cassandra is a non-relational database being classified as a NoSQL database system. Instead of storing the data in a row-oriented manner, Cassandra is a column-based storing system. Therefore, it comes with a different approach. While PostgreSQL and MongoDB work with databases, Cassandra uses keyspaces consisting of a set of column families. Listing 5.7 shows how the keyspace "iowdata" is created by using the `CREATE KEYSPACE`-command. 'SimpleStrategy' was selected as class for the keyspace and the replication factor is set to '1'. This means that the keyspace is created on a single node cluster and assigns the same replication factor to the entire cluster. In order to enter the keyspace the USE-command is used like depicted in Listing 5.8.

```
1 CREATE KEYSPACE iowdata WITH replication = {
2 'class': 'SimpleStrategy',
3 'replication_factor': '1'
4 };
```

Listing 5.7: Creating a Keyspace in Cassandra

```
1 USE iowdata;
```

Listing 5.8: Entering the Keyspace in Cassandra

Subsequently, a column family is created by using the `CREATE TABLE`-command. In contrast to SQL-based table creation, the primary key consists of a set of partition and clustering keys. **profile_id** is defined as the partition key while **ts** is set as clustering key. This ensures that data within a partition later can be ordered by timestamp. In addition, it is defined in line 12 that the order should be ascending. Finally, the data is inserted similarly to SQL with the only difference, that each key-value-pair needs to be inserted individually (like presented in Listing 5.10). By applying the `BATCH`-environment, all `INSERT INTO`-commands can be performed in one unit.

```
1  CREATE TABLE sampledata
2  (
3      profile_id int,
4      deployment_id int,
5      ts timestamp,
6      conductivity_value float,
7      concoeff_1 float,
8      concoeff_2 float,
9      concoeff_3 float,
10     concoeff_4 float,
11     PRIMARY KEY (profile_id, ts)
12 ) WITH CLUSTERING ORDER BY (ts ASC);
```

Listing 5.9: Column Family Model in Cassandra

```
1 BEGIN BATCH
2         INSERT INTO sampledata(profile_id, deployment_id, ts, conductivity_value,
              concoeff_1, concoeff_2, concoeff_3, concoeff_4)
3         VALUES (201, 8, '2009-08-22 14:07:22.643', 11504, 0.0922283727, 0.00123643726,
              0.00000000000999383726, 0.000000000000000120393922);
4         INSERT INTO sampledata(profile_id, deployment_id, ts, conductivity_value,
              concoeff_1, concoeff_2, concoeff_3, concoeff_4)
5         VALUES (434, 12, '2014-04-01 23:59:59.892', 14511, 0.0886499939, 0.00101438668,
              0.0000000000935514578, 0.000000000000000110249781);
6 APPLY BATCH;
```

Listing 5.10: Creating Key-Value-Pairs in Cassandra

MongoDB

Like Cassandra, MongoDB is classified as a non-relational database and therefore comes with a different approach compared to PostgreSQL. In contrast to Cassandra and SQL-based databases, MongoDB does not store data in column families or relations, but in collections of documents. Like key-value pairs in Cassandra, documents do not have to follow defined constraints. They are created on the fly and do not need to contain entries for every field inside the collection. Due to ensure a more significant comparison of the used database systems, it is assumed that each document inserted consists of the same fields.

Before data can be incorporated, a database and a collection within the database need to be created. Listing 5.11 and Listing 5.12 demonstrate how this is realized in the Mongo shell. The new database "iowdata" is created by simply using the command *use* < *database name* >. In contrast to Cassandra, the user do not need to enter the database. If the user later want to switch to a different database he simply uses the use-command again. Next, the collection "sampledata" is created by using the createCollection-command.

```
use iowdata
```

Listing 5.11: Creating a New Database in MongoDB

```
iowdata.createCollection(sampledata)
```

Listing 5.12: Creating a Collection in MongoDB

Different collections can be addressed without switching from one to another. This applies for all basic operations such as inserting, updating and deleting data. Listing 5.13 displays an example of how the underlying data model is inserted in MongoDB by appending the insert-operator to the collection name. The data is structured as documents and the documents are stored in JSON. Although it is not necessary, a field type was defined for every field to ensure a higher comparability of the model. In contrast to Cassandra and PostgreSQL, MongoDB does not define a column (here: field) as primary key. Rather, an object ID ($oid) is added automatically to each document to ensure uniqueness. Similarly to CQL, each document need to be inserted individually. After all entries of the exemplary data model have been entered, the collection was completed.

```
db.sampledata.insert([
{
    "_id": {"$oid": "5e2eb07d9226ce4108941d10"},
    "profile_id": {"$numberLong": "201"},
    "deployment_id": {"$numberLong": "8"},
    "ts": {"$date": {"$numberLong": "1250942842643"}},
    "conductivity_value": {"$numberDouble": "11504"},
    "concoeff_1": {"$numberDouble": "0.0922283727"},
    "concoeff_2": {"$numberDouble": "0.00123643726"},
    "concoeff_3": {"$numberDouble": "9.99383726e-12"},
    "concoeff_4": {"$numberDouble": "1.20393922e-16"}
}}])
```

Listing 5.13: Collection Model in MongoDB

This section has shown that creating the same exemplary data model in three different systems can be highly individual. Furthermore it pointed out that creating this model is considerably different from the vector-based approach explored in section 5.1. Apart from differences in the data model, the systems can vary heavily in syntax. Thus depending on the requirements, it can be less or more easy to create

the data model and the user needs to specify his needs ahead of time. In section 5.3 and section 5.4 it will be further elaborated on this by exploring how sorting and polynomial evaluation can be transformed to the used database systems. Thereby, section 5.3 will focus on sorting the data model by timestamp while section 5.4 will explore how conductivity can be computed by performing polynomial evaluation.

5.3. Database-Supported Sorting

The following section explores which queries are used in order to sort the exemplary data model by timestamp. Furthermore, it will be shown which output the different queries produce. As mentioned in section 4.3, sorting has achieved different degrees of feasibility for the used database systems. Table 8 once again displays the achieved degrees. Whereas sorting can be realized by simply using prefabricated functions and operators in PostgreSQL and MongoDB, it is comparably different to order column families in Cassandra. Therefore dependent on the data model and the query language syntax, different databases will produce different output. In this section it will be illustrated in which way the outputs differ from each other and how they refer to the reference model's results. Furthermore, measured execution times for sorting the exemplary data model by timestamp will be displayed for all used database systems and the Matlab reference model.

FUNCTIONALITY	**SQL-Standard**	**PostgreSQL**	**Cassandra**	**MongoDB**
Sorting	3	3	2	3

Table 8.: Sorting: Comparison across Database-Systems based on Degree of Feasibility

PostgreSQL

Since PostgreSQL relies on a relational database approach, relations can be ordered by any arbitrary column or even by multiple columns at the same time. Listing 5.14 demonstrates how the exemplary data model is sorted in order to reproduce the results generated by the vector-based Matlab approach in section 5.1. As already explained PostgreSQL basically queries the stored data by using the `SELECT`-environment. By applying the `ORDER BY`-clause inside the `SELECT`-statement environment, the entire table can be sorted by **ts**. By setting using $*$, it is ensured that sorting will be applied for all tuples of the target table. As a result the entire related will be ordered like specified.

```
1 SELECT * FROM sampledata ORDER BY ts;
```

Listing 5.14: Sorting by Timestamp in PostgreSQL

Table 9 shows the output produced by the applied `SELECT`-statement. It can be observed that the relation is now ordered by timestamp. As a result, the problem of profile 434 exceeding the date is solved and all tuples of the relation "sampledata" are ordered chronologically. Thereby, the remaining columns are ordered based on the sorting index of the **ts**-column which means that every tuple of the relation retain its related values. Compared to the reference model , both approaches lead to the same results (see Table 7. Hence, PostgreSQL was able to reproduce the output of the Matlab-analysis exactly as expected.

	profile_id integer	deployment_id integer	ts [PK] timestamp without time zone	conductivity_value real	concoeff_1 real	concoeff_2 real	concoeff_3 real	concoeff_4 real
1	201	8	2009-08-22 14:07:22.643	11504	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
2	201	8	2009-08-22 14:07:22.869	11502	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
3	201	8	2009-08-22 14:07:23.125	11502	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
4	201	8	2009-08-22 14:07:23.407	11500	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
5	201	8	2009-08-22 14:07:23.665	11499	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
6	201	8	2009-08-22 14:07:23.924	11498	0.092228375	0.0012364372	9.993837e-12	1.2039393e-16
7	434	12	2014-04-01 23:59:58.623	14514	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
8	434	12	2014-04-01 23:59:58.834	14514	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
9	434	12	2014-04-01 23:59:59.031	14513	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
10	434	12	2014-04-01 23:59:59.277	14512	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
11	434	12	2014-04-01 23:59:59.606	14512	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
12	434	12	2014-04-01 23:59:59.892	14511	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
13	434	12	2014-04-02 00:00:00.21	14510	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
14	434	12	2014-04-02 00:00:00.487	14509	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
15	434	12	2014-04-02 00:00:00.992	14508	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
16	434	12	2014-04-02 00:00:01.249	14509	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
17	435	12	2014-04-02 00:00:01.553	14510	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16
18	435	12	2014-04-02 00:00:01.791	14510	0.088649996	0.0010143867	9.355146e-12	1.1024978e-16

Table 9.: Sorted Output in PostgreSQL

Cassandra

In contrast to SQL-based databases, it is simply not possible in Cassandra to order an entire column family by an arbitrary column only by querying and without considering the sorting structure ahead of time before the data model is constructed. This is because Cassandra uses a set of partition and clustering keys instead of the primary key applied in SQL. In this data model, **profile_id** is defined as partition key whereas **ts** constitutes the clustering key within each partition. Hence, data will be ordered automatically by the specified clustering key in every single partition.

Listing 5.15 presents the query that is used to sort the data in the exemplary data model. In contrast to PostgreSQL, the query requires no `ORDER BY`-clause inside the `SELECT`-statement environment because the data is ordered automatically by the defined clustering-key. The `ORDER BY`-clause would only be used in order to reverse the sorting direction of the cluster. Similarly to the query applied in PostgreSQL, every key-value-pair of the entire target column family "sampledata" can be addressed by using the $*$-operator. Listing 5.15 demonstrates that the same query that was implemented in PostgreSQL can be reproduced with less code. The drawback is that the user needs to specify a definite data model ahead of time and therefore exactly needs to know which queries will be performed.

```
1 SELECT * FROM sampledata;
```

Listing 5.15: Automated Sorting by Clustering Key in Cassandra

Table 10 screens the output produced by executing the sorting query in the Cassandra shell. Unlike the results printed in the PostgreSQL environment, the columns changed their order. Thereby, the partition key (**profile_id**) ranks first while the clustering key (**ts**) ranks second. The remaining columns inside the column family kept their order whereas the only column that previously was positioned in between the partition key and clustering key (**deployment_id**) was attached to the end and now constitutes the final column. The displayed output also demonstrates that Cassandra achieved the same results like PostgreSQL and the reference model exactly as expected (see Table 7. Furthermore, Cassandra uses less code by defining the sorting order while specifying the data model of the column family.

profile_id	ts	concoeff_1	concoeff_2	concoeff_3	concoeff_4	conductivity_value	deployment_id
201	2009-08-22 12:07:22+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11504	8
201	2009-08-22 12:07:22+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11502	8
201	2009-08-22 12:07:23+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11502	8
201	2009-08-22 12:07:23+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11500	8
201	2009-08-22 12:07:23+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11499	8
201	2009-08-22 12:07:23+0000	0.092228	0.001236	9.9938e-12	1.2039e-16	11498	8
434	2014-04-01 21:59:58+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14514	12
434	2014-04-01 21:59:58+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14514	12
434	2014-04-01 21:59:59+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14513	12
434	2014-04-01 21:59:59+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14512	12
434	2014-04-01 21:59:59+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14512	12
434	2014-04-01 21:59:59+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14511	12
434	2014-04-01 22:00:00+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14510	12
434	2014-04-01 22:00:00+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14509	12
434	2014-04-01 22:00:00+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14508	12
434	2014-04-01 22:00:01+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14509	12
435	2014-04-01 22:00:01+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14510	12
435	2014-04-01 22:00:01+0000	0.08865	0.001014	9.3551e-12	1.1025e-16	14510	12

Table 10.: Sorted Output in Cassandra

MongoDB

MongoDB uses an entirely different approach to organize and sort its data. Since MongoDB is neither a row-based nor a column-based database system, it relies on documents and collections and provides a totally different querying language compared to SQL and CQL. Thus, similar results are achieved by applying different statements and producing a differently structured output. Like mentioned in section 4.3, MongoDB provides the *sort*-operator that works analogously to SQL's `SELECT`-statement and is simply appended to the *find*()-command. The operator allows to order by any arbitrary column and even by multiple columns.

Listing 5.16 depicts the query that was performed to sort the documents stored in the database. In this case, the collection "sampledata" is ordered by the **ts**-field. By setting the order option to "1" it is ensured that the documents will be displayed in ascending order. Because MongoDB works with document instead of table environments, it can be sometimes difficult to review the printed output. Therefore, the *pretty*()-command is attached to print the output lucidly. It configures the cursor and adds space in order to display the results in an easy-to-read-format.

```
1 db.sampledata.find().sort({ts:1}).pretty()
```

Listing 5.16: Sorting by Timestamp in MongoDB

Although the queries of MongoDB can be often used almost analogously to the related SQL-queries, the output looks enormously different. Listing 5.17 screens an extract of the results produced by executing the shown query in the Mongo shell. Thereby, two instances are shown demonstrating that exceeding the date is not an issue any more. The first instance identified by the `ObjectId "5e2eb41a02a5b0410826acd7"` displays the last record of profile 434 on the first of April in 2014. The second instance identified by the `ObjectId "5e2eb36302a5b0410826accc"` displays the first record of profile 434 on the second of April in 2014.

Since the data is stored in JSON format, the output is printed in JSON, as well. Thereby, each JSON document retains its structure. The only difference is that the documents now are ordered by timestamp. Therefore, MongoDB basically achieves the same results like PostgreSQL, Cassandra and the reference model implemented in Matlab (see Table 7). Although the output design relies on the structure of the stored documents and therefore looks a little bit different from table-oriented structures, the results are the same as expected.

```
}{
        "_id" : ObjectId("5e2eb41a02a5b0410826acd7"),
        "profile_id" : NumberLong(434),
        "deployment_id" : NumberLong(12),
        "ts" : ISODate("2014-04-01T21:59:59.892Z"),
        "conductivity_value" : 14511,
        "concoeff_1" : 0.0886499939,
        "concoeff_2" : 0.00101438668,
        "concoeff_3" : 9.35514578e-12,
        "concoeff_4" : 1.10249781e-16
}{
        "_id" : ObjectId("5e2eb36302a5b0410826accc"),
        "profile_id" : NumberLong(434),
        "deployment_id" : NumberLong(12),
        "ts" : ISODate("2014-04-01T22:00:00.210Z"),
        "conductivity_value" : 14510,
        "concoeff_1" : 0.0886499939,
        "concoeff_2" : 0.00101438668,
        "concoeff_3" : 9.35514578e-12,
        "concoeff_4" : 1.10249781e-16
}{
```

Listing 5.17: Sorted Output in MongoDB (Extract)

Execution Times and Summary

Although the exemplary data model is quite small, execution times were measured for all performed sorting algorithms on a hard disk environment. This includes the three used database systems as well as Matlab as a reference system. Thereby, the execution time was measured in milliseconds. The executions have been measured ten times and afterwards an average was calculated for each of the involved systems. Table 11 screens the measured results. It can be concluded that PostgreSQL as due to its row storage and concurrency control is 80 times slower than the schema-less NoSQL databases. For MongoDB there even could not been measured any value higher than zero, since it operates within the microsecond range. Furthermore, Matlab was faster than PostgreSQL as well, but slower than Cassandra and MongoDB. In this case, the differences in speed can be considered negligible due to the small data set size. But when it comes to massive data, this could mean a potential issue for relational DBMS and Matlab.

Number of Execution	Matlab	PostgreSQL	Cassandra	MongoDB
1	7.2	82	0.002	0
2	4.2	78	0.002	0
3	8.8	78	0.003	0
4	5.2	81	0.002	0
5	8.9	74	0.002	0
6	4.0	83	0.002	0
7	3.8	82	0.002	0
8	5.6	114	0.003	0
9	5.0	79	0.002	0
10	3.7	92	0.002	0
∅	5.6	84.3	0.002	0.0

Table 11.: Execution Times for Sorting in Milliseconds

In conclusion, all used databases met the expectations and produced the same results like the reference model when it comes to sort the exemplary data model by timestamp. Thereby, each database provides its unique technique and the output appearance can vary from system to system. Moreover, NoSQL

databases were faster than Matlab and PostgreSQL. But substantively, all approaches led to the same results. In section 5.4 it will be explored how conductivity can be computed based on polynomial evaluation. Furthermore, it will be analyzed whether all databases are again able to meet the expectations by reproducing the same output like the reference model.

5.4. Database-Supported Polynomial Evaluation for Conductivity

As a next step, it will be explored which queries were applied to compute the actual conductivity values by polynomial evaluation based on 16-bit-integers. Therefore, the following section will describe how polynomial evaluation performed by the Matlab reference model can be transformed to the used database systems. Furthermore, the output produced by the four different systems will be displayed and compared to each other. Table 12 once again displays the achieved degrees of feasibility. Like depicted in section 4.3 none of the database systems is able to transform this functionality in a one-to-one-manner or by a prefabricated function. Nevertheless, polynomial evaluation can be performed in a different way with reasonable effort. Although polynomial evaluation achieves the same level of technical feasibility on any database system, every database provides an individual technique to realize the transformation. Hence, it is shown what techniques are applied and how they differ from each other. Additionally, the techniques will be compared in terms of the measured execution time.

FUNCTIONALITY	**SQL-Standard**	**PostgreSQL**	**Cassandra**	**MongoDB**
Polynomial Evaluation	2	2	2	2

Table 12.: Polynomial Evaluation: Comparison across Database-Systems based on Degree of Feasibility

PostgreSQL

In PostgreSQL, there are two different techniques that could be applied. The polynomial evaluation can be either performed by using a scalar expression inside the `SELECT`-statement or by implementing an UDF in an internal or external programming language. Listing 5.18 shows how the evaluation is transformed by implementing the polynomial as a scalar expression. Thereby, the expression is simply implemented as a sequence of addressed column names and mathematical operators within the `SELECT`-environment and saved as "conductivity". The results are then sorted by timestamp by applying the `ORDER BY`-statement. The advantage of this method is that the user does not need to specify an UDF and the operation can be performed directly on a database level.

```
1 SELECT sampledata.concoeff_1 + sampledata.conductivity_value * (sampledata.concoeff_2 +
      sampledata.conductivity_value*(sampledata.concoeff_3+sampledata.conductivity_value*
      sampledata.concoeff_4))
2 AS conductivity
3 FROM sampledata
4 ORDER BY sampledata.ts;
```

Listing 5.18: Polynomial Evaluation in PostgreSQL (Scalar Expression)

Instead of scalar expressions, the user can also define an UDF and later apply it on a target relation. Listing 5.19 demonstrates how this was realized on the exemplary data model. Firstly, an UDF called "polyval" was specified by using the `CREATE OR REPLACE FUNCTION`-command. Thereby, the `REPLACE`-statement ensures that all functions previously implemented having the same name will be overwritten to avoid functional redundancy. In the parenthesis, input variables are specified. The output is specified

right after the RETURNS-statement. In this case five floating numbers are used as input and one floating number as output. Between the BEGIN-statement and the END-statement the UDF is implemented based on the language that was specified by the LANGUAGE-statement. Since the polynomial is relatively less sophisticated, the procedural language of PostgreSQL, "plpgsql", could be used.

Lines eight, nine and ten demonstrate how the UDF is used inside the SELECT-statement. In contrast to the scalar expression, the query is shorter, since each relevant column can be typed simply inside the *polyval*-function. The output is once again saved as "conductivity" and sorted by timestamp by using the ORDER BY-statement. The drawback of this method is, that the user needs to know how the UDF is specified, which can be an issue in terms of data provenance. The advantage of this method is that the UDF can be applied as often as needed and thereby shortens query expressions.

```
1  CREATE OR REPLACE FUNCTION polyval(a real,b real,c real,d real, x real)
2  RETURNS real AS $$
3  BEGIN
4          RETURN a + x * (b + x*(c+x*d));
5  END; $$
6  LANGUAGE plpgsql;
7
8  SELECT polyval(sampledata.concoeff_1,sampledata.concoeff_2,sampledata.concoeff_3,
       sampledata.concoeff_4,sampledata.conductivity_value) AS conductivity
9  FROM sampledata
10 ORDER BY sampledata.ts;
```

Listing 5.19: Polynomial Evalutation in PostgreSQL (UDF)

Cassandra

In contrast to SQL-based databases, Cassandra's CQL does not allow to use scalar expressions directly within the SELECT-statement. Therefore, implementing an UDF remains the only possibility for realizing a transformation of a polynomial evaluation. Procedurally, implementing an UDF in CQL works almost similarly like in SQL. Listing 5.20 illustrates how polynomial evaluation is performed by creating an UDF in Cassandra. The major difference is that Cassandra has no own procedural language, and therefore UDFs are implemented in Java or JavaScript by default. Furthermore, the possibility of implementing UDFs needs to be enabled in the Cassandra YAML-file that manages general settings of the server and additional programming languages need to be installed on the users own. To update the settings, the Cassandra server needs to be restarted.

In this case, the UDF is implemented by the systems default language right after the AS-statement. Since the default language is Java, the UDF is implemented by using the familiar `return`-expression followed by a sequence of variables and mathematical operators. Furthermore, behaviour on invocation with zero values must be defined for each UDF. This is realized by using the `RETURNS NULL ON NULL INPUT`-statement declaring that the function will always return null if any of the input arguments is null. Apart from that, the UDF is implemented and deployed like in SQL. This means that the UDF is created by using the `CREATE OR REPLACE FUNCTION`-environment and specified for producing one floating number from five inputted floating numbers by executing the defined scalar expression. Additionally, the UDF is applied within the SELECT-statement and saved as "conductivity". Similarly to the implementation of sorting in section 5.3, no `ORDER BY`-statement is required, since Cassandra orders each outputted partition automatically by the defined clustering key.

```
1 CREATE OR REPLACE FUNCTION polyval(a float,b float,c float,d float, x float)
2 RETURNS NULL ON NULL INPUT
3 RETURNS float
4 LANGUAGE java
5 AS 'return a + x * (b + x*(c+x*d));';
6
7 SELECT polyval(concoeff_1,concoeff_2,concoeff_3,concoeff_4,conductivity_value) AS
      conductivity
8 FROM sampledata;
```

Listing 5.20: Polynomial Evaluation in Cassandra

MongoDB

Similarly to the realization of sorting the data model by timestamp, MongoDB provides a different approach relying on a totally different querying language syntax. Like mentioned in section 4.3, MongoDB offers the *forEach*()-operator for iterating through a collection of documents. Listing 5.21 shows how polynomial evaluation for conductivity is realized for each document by iterating over the collection "sampledata". The operate is simply appended to a sequence of statements including the *find*-statement and the *sort*()-operator. Since MongoDB relies on JavaScript and JSON, a JavaScript function is used to compute the conductivity values by writing a scalar expression inside the *function*(){}-environment. Thereby, it is specified, that the function should be applied on the document type (doc) and print the result of the implemented scalar expression for each document inside the collection *print*().

```
1 db.sampledata.find().sort({ts:1}).forEach(function(doc){print(doc.concoeff_1+doc.
      conductivity_value*(doc.concoeff_2+doc.conductivity_value*(doc.concoeff_3+doc.
      conductivity_value*doc.concoeff_4)))})
```

Listing 5.21: Polynomial Evaluation in MongoDB

Results and Summary

All in all, it could be shown that there are some differences in the way the transformation was realized. Although all databases reached the same degree of technical feasibility concerning polynomial evaluation, they provide several techniques to achieve it. These techniques for computing conductivity based on polynomial evaluation split into direct querying of scalar expressions, iterative querying or UDFs. In the case of PostgreSQL both a direct querying approach and a UDF were used. In Cassandra, polynomial evaluation was implemented as UDF as well, but in a different procedural language. In MongoDB, the functionality was realized by iterative querying over a collection of documents.

Furthermore, some databases provide an individual procedural language while others rely on external languages. For instance, PostgreSQL provides an individual procedural language for implementing UDFs whereas Apache's Cassandra relies on an external procedural language. But despite the differences in realization and the applied programming language, all databases were able to realize polynomial evaluation with reasonable effort.

Table 13 displays the computed conductivity values printed by the used database systems and the reference model. Although the output sometimes is rounded differently dependent on the underlying system, all databases printed the same results like the reference model. Thus, there are no differences between relational and non-relational systems in concerns of the printed results and each program including PostgreSQL, Cassandra and MongoDB delivered exactly the values that were expected.

Matlab	PostgreSQL	Cassandra	MongoDB
14.3177	14.317707	14.31771	14.317708511548304
14.3152	14.315234	14.31523	14.315235081609439
14.3152	14.315234	14.31523	14.315235081609439
14.3128	14.312761	14.31276	14.312761651783754
14.3115	14.311524	14.31152	14.311524936913358
14.3103	14.310287	14.31029	14.310288222071254
14.8138	14.813766	14.81377	14.813766071582627
14.8138	14.813766	14.81377	14.813766071582627
14.8128	14.812751	14.81275	14.812751343681212
14.8117	14.811737	14.81174	14.811736615808107
14.8117	14.811737	14.81174	14.811736615808107
14.8107	14.810722	14.81072	14.810721887963313
14.8097	14.809706	14.80971	14.809707160146829
14.8087	14.808691	14.80869	14.808692432358653
14.8077	14.807676	14.80768	14.807677704598785
14.8087	14.808691	14.80869	14.808692432358653
14.8097	14.809706	14.80971	14.809707160146829
14.8097	14.809706	14.80971	14.809707160146829

Table 13.: Polynomial Evaluation for Conductivity: Output of all Systems

Table 14 presents the measured execution time in milliseconds for each of the performed evaluations. Similarly to section 5.3, each execution has been measured ten times on a hard disk environment and an average was calculated afterwards (twice for PostgreSQL: scalar expression (SE) and UDF). The values show that Matlab and MongoDB performed 80 times (or in case of UDF 90 times) faster than PostgreSQL and 110 times faster than Cassandra. Generally, databases that used UDFs were slower than the remaining systems which is because they first need to save a function before the query can be executed. Furthermore, PostgreSQL performed slightly better than Cassandra in polynomial evaluation although both systems applied UDFs. This is probably because PostgreSQL provides an internal procedural language and Cassandra relies on external languages. Another interesting point is that MongoDB once again performs in the microsecond range and therefore only zero values could be measured.

Number of Execution	Matlab	PostgreSQL		Cassandra	MongoDB
		SE	UDF		
1	0.5	81	84	111	0
2	0.1	74	86	122	0
3	0.4	73	95	103	0
4	0.2	83	80	136	0
5	0.7	80	126	115	0
6	0.1	76	76	108	0
7	0.1	87	103	103	0
8	0.1	84	93	105	0
9	0.1	73	92	101	0
10	0.1	91	84	104	0
∅	0.2	80.2	91.9	110.8	0.0

Table 14.: Execution Times for Polynomial Evaluation in Milliseconds

In conclusion, all databases met the expectations and produced the same results like the reference model in terms of evaluating the polynomial for conductivity. Thereby, each database provides its unique technique, but the output appearance is equal. In chapter 6, these results will be discussed critically and current opinions on the differences between SQL and NoSQL databases will be explored.

6. Discussion

The results have shown that it is possible to reproduce parts of the Matlab analysis by applying database-supported evaluations. Although the examples clearly demonstrate how Matlab programs can be transformed almost in a one-to-one-manner and sometimes with more efficient performance, there is still a lot of work and research that needs to be done for realizing a database-supported transformation of the entire Matlab analysis. Due to the rather representative characteristics of the constructed exemplary data model and other capacities, there are some points that need to be criticized or require further examination. Therefore, the following chapter will explore occurred problems as well as potential issues. Additionally, current opinions on the differences between traditional RDBMS and novel NoSQL will be discussed. Moreover, it will be stated which fields still require further research and how these fields should be treated.

The process of transforming Matlab based analysis to database-supported evaluations offered some difficulties that either needed to be faced or affected the project's results. First of all, the Matlab code is highly sophisticated and consists of over thousands lines of code. On top of that, there is a wide range of researchers that did work on the files. Consequently, a big majority of the people working with the files are only users and therefore are only sparsely familiar with the details of the code. Thus, it can be very difficult for an outsider and even for the user to retrace any step in detail. Due to the high complexity of the Matlab-analysis, further research needs to be done on how a transformation of the entire original Matlab analysis can be realized.

Apart from that, it has sometimes been an issue to work with NoSQL databases, since they are far less developed than the relational dinosaur-databases. They were less user-friendly and difficult to install compared to the relational database. Furthermore, PostgreSQL's database environment, pgAdmin, was easier to use and provided greater functionality than the NoSQL Shells or MongoDB Compass. For instance, it was not possible to copy code into the MongoDB shell or to perform complex queries inside the environment of MongoDB Compass. In addition, Cassandra disables UDFs by default so that is was necessary to change the settings of the server before the data could be analyzed. But the main issue remains the highly sophisticated Matlab-analysis.

Therefore, the data model that was used in this paper provides only representative character. This means that not all functionalities were included and the data model was constructed manually. Hence, the original Matlab analysis was strongly simplified by constructing a reference model in Matlab. Furthermore, there is no underlying database system yet developed for the purposes of the GODESS data. This would require extensive effort and offers enough tasks for a standalone project. Hence, each data model was created manually for each platform and did not use a unified TXT-file as input which potentially makes the results of each platform difficult to compare.

Thereby, the data model includes only parts of the data included in the CTD and the XML-file. Data from the TriOS and Nortek files were completely excluded. Another point of criticism is that data from the CTD and XML files were combined in a single representation and not in two standalone tables, column-families or collections which could have a massive impact on the results. Due to the great complexity of

the original scripts and the enormous simplification of the reference Matlab model it is imaginable that some functionalities included in the original scripts were overlooked and therefore were not considered while analyzing the feasibility of the transformation. On top of that, UDFs provide almost endless possibilities to implement anything required for the user's data analysis. Basically, any potential problem can be solved by UDFs relying on more or less sophisticated algorithms. This involves even functionalities that were assigned to the lowest defined degree of technical feasibility including multiple linear regression and graph plotting. Therefore, it was sometimes difficult to distinguish which functionalities would be transformable with reasonable effort and which not.

It is also imaginable, that PostgreSQL, Cassandra or MongoDB do not represent the platform to be the most suitable of their categories. Probably, different relational database platforms, column stores or document stores would have suited the requirements better and therefore covered a wider range of functionalities than need to be transformed. Due to the different schemata applied by the employed databases, it is also difficult to compare the databases with Matlab or even each other. In terms of comparative criteria, only execution time was measured and therefore constitutes no fully representative parameter. Furthermore, other criteria such as memory usage need to be considered and there is required further research to analyze the performance of the employed platforms.

Another aspect that needs to be discussed is how suitable relational and non-relational databases are in general to face the problems of a Matlab transformation and what issues remain despite their capabilities. Since relational DBMS remain the main data management technology despite the trend of NoSQL-waves [NPP13], it is a common opinion that NoSQL systems will stay nothing but a trend appearance. Supporters of this idea state that NoSQL databases will suffer from the same fate like object-oriented databases whose features were took over by relational databases and significantly decreased the number of users of such databases. [Ord13] The fact that NoSQL still lags behind the relational grandpa in terms of numbers of users [NPP13] could already be a warning symbol.

One reason for this fact is that there is no querying language in the NoSQL area being able to span multiple data models at the same time, because each class of NoSQL database is designed for a specific purpose. [KR13] Thus, some researchers claim there is a need for a common NoSQL query language which can be used for all NoSQL databases, if the NoSQL movement wants to survive. [NPP13] Therefore, Unstructured Query Language (UnQL) constitutes the most promising but still immature approach. [KR13] Developed by the creators of SQLite and CouchDB, it is one collective effort to bring a familiar and standardized data definition and manipulation language relying on SQL like syntax, open-source development and features that combine structures from both relational and non-relational systems. [NPP13] If a uniformed language such as UnQL has the chance to mature before the NoSQL movement is possibly devoured by the relational dinosaur, it could probably be a game changer.

Apart from the competition between relational and non-relational databases, there is the question whether relational databases are able to face the same problems like statistical languages. Critics claim that RDBMSs could be beaten by specialized architectures such a scientific and intelligence databases or array storage engines like Matlab. [SK11] They argue that the integration of mathematical packages or scientific computing programming languages like R and Matlab into databases and systems relying on MapReduce still remains limited. [Ord13] Even this case has shown that transforming statistical analysis to database-supported evaluations brings along various difficulties arising out of the immatureness of the databases' statistical libraries or lacks in dealing with data structures like arrays and matrices. On the other hand, the problem of integrating statistical and machine learning methods and models with RDBMSs has received moderate attention. [Ord13] There is still a lot of research to do on these issues, but it can be clearly stated that statistical methods as well as complex data structures like matrices will be

more and more integrated into common database systems. The past has shown over and over again that relational databases were able to adapt techniques developed by other systems. Maybe it is only a matter of time until they manage integrating the features that make NoSQL databases and statistical languages still a better fit for specific purposes, as well.

Since this experiment has shown that neither a relational nor a NoSQL database was capable to transform the entire Matlab analysis in a 1-to-1-manner with the same level of effort, I suggest to not view the different platforms as a replacement for each other, but as a complementary product. Right at the moment, the community agrees that SQL and NoSQL cannot be used interchangeable for solving any type of problem but the user should rather choose between the two types of databases for a given instance.[TB11] To my mind, the same applies to the relationship of Matlab and database applications in general. Like the illustration has indicated, there are still parts of the analysis that are more efficient to perform in Matlab. Therefore, it would be a great idea to store the data on a database server and to perform feasible functions on a database level. In addition, functions requiring too extensive effort, should still be performed in Matlab. Thus, it cannot be stated which database consitutes the most suitable standalone solution.

There is rather required a holistic approach considering the strengths of both sides including Matlab and the database server. The more features of statistical languages can be efficiently transformed to database-supported evaluation, the more efficient this hybrid model will generally perform. To increase this rate, it is necessary to study the integration of statistical methods and matrix computations with database systems in more depth and to extend DBMSs with matrix data types, matrix operators and linear algebra methods.[Ord13] Additionally, databases should incorporate array and matrix storage and be extended to include mathematical libraries.[OBLB15] Furthermore in concerns of UDFs, query optimization techniques need to be adapted. Since UDFs constitute a great possibility for integration (as demonstrated in this paper) but work in main memory, there is a strong need of optimization. Otherwise, they cannot call relational operators, are fed by table scan operators and therefore impair execution time and memory usage dramatically.[Ord13]

Generally, more and more processes are becoming digitized and users often can be overstrained by the wide range of database systems to choose from. Therefore, it needs to become a common business that users analyze their needs. Furthermore, users needs to became better educated to decide more efficiently between the variability of available databases and to draw conclusions about the necessity of employing a database system for their needs. If users decide upon launching database systems they should consider different criteria. This involves their businesss model, the demand for ACID transactions and potential costs.[HHLD11] Moreover, potential querying language, involved features and the strenghs and weaknesses of the underlying data model should be considered.[NPP13]

As this chapter has pointed out, there is a lot of research to be performed in concerns of different aspects when it comes to statistical analysis to be transformed on a database server. This includes the further development of databases in terms of matrix operations, array storage and mathematical packages as well as to find ways of applying different platforms complementary to achieve more efficient performance. Furthermore, it was stated that there is a bunch of difficulties that need to be faced in order to transform the Matlab analysis to database-supported evaluations and that the employed data model as well as the reference model are capable of future improvement. Nevertheless, both models can be viewed as strongly represent illustrations underlying the mentioned needs for future research.

7. Conclusion

This thesis examined how Matlab-analysis that are performing strongly memory consuming algorithms can be potentially transformed into database-supported evaluations in SQL and NoSQL systems. Thereby, this work focuses on the Matlab analysis that is performed by researchers of the IOW working with highly sensitive scientific sensor-data gathered by the GODESS in the Godland Basin. Due to the highly sophisticated analysis, a simplified reference model was developed to illustrate the transformation. Furthermore, an exemplary data model was constructed to represent the broad variety of semi-structured data that is applied in the project at the IOW.

Based on the gathered data and the Matlab scripts it was analyzed which functionalities are included in the performed Matlab analysis. Thereby, ten basic functionalities were found. Out of these functionalities, two examples were included in the reference model. This involved sorting and polynomial evaluation for computing conductivity. The functionalities were transformed to various database systems. Based on the latest state of art and the data properties, three individual databases were chosen. As a relational DBMS, PostgreSQL has been picked. Furthermore, two non-relational databases were chosen. This includes Apache's Cassandra as column store and MongoDB as a representative instance for document stores.

Although each database system relies on a different data model and therefore different techniques of implementation needed to be realized, all databases basically produced the same input as the reference model. Thereby, NoSQL databases performed faster than relational databases and databases applying UDFs performed slower than other databases. Generally, it could be shown that no database is capable to transform the entirety of perform Matlab functionalities on its own, but relational databases seem to cover most functionalities out of all databases considered.

Therefore, it is recommended to launch a holistic approach that combines the benefits of both Matlab and relational databases. The approach should store the data on a database server and perform all functionalities that can be transformed with reasonable effort on a database level. Other functionalities causing too extensive effort such as multiple linear regression and the plotting of scientific graphs should be exported to Matlab. This will reduce memory usage at the one side but keep efficiency on the other side. Furthermore, further research needs to be done on how strongly specialized architectures like statistical methods and array storage or dynamic schemata like provided by NoSQL databases can be transformed to relational databases in order to make these databases catch up on the recent needs of the digitized world.

A. Appendix

This Bachelor Thesis will be submitted together with a flash drive containing the following contents:

- The Bachelor Thesis as PDF-File
- The Bachelor Thesis as LaTeX-Directory
- An alphabetical List of PDF-Files of the used References with their used Acronyms
- The used Source-Code
  (1) The Matlab-Script
  (2) The SQL-Code
  (3) The CQL-Code
  (4) The MQL-Code
- The Original Matlab-Scripts
  (a) The Main Script
  (b) The Script for Calculating Practical Salinity
  (c) The Script for Calculating Absolute Salinity
- The Original Data
  (a) The CTD-File
  (b) The TriOS-File
  (c) The Nortek-File
  (d) The Metadata-File

Bibliography

[AB13] Abramova, V. ; Bernardino, J.: NoSQL Databases: MongoDB vs Cassandra. In: *Proceedings of the international Conference on Computer Science and Software Engineering* ACM, 2013, S. 14–22

[Apa14] Apache: Apache Cassandra. In: *Website. Available online at www.planetcassandra.org/what-is-apache-cassandra* 13 (2014)

[BKM+17] Bruder, I. ; Klettke, M. ; Möller, M. L. ; Meyer, Fr. ; Heuer, A. ; Jürgensmann, S. ; Feistel, S.: Daten wie Sand am Meer–Datenerhebung,-strukturierung,-management und Data Provenance für die Ostseeforschung. In: *Datenbank-Spektrum* 17 (2017), Nr. 2, S. 183–196

[Bre00] Brewer, E. A.: Towards Robust Distributed Systems. In: *PODC* Bd. 7, 2000

[Cod70] Codd, E. F.: A Relational Model of Data for large shared Data Banks. In: *Communications of the ACM* 13 (1970), Nr. 6, S. 377–387

[CSD11] Cuzzocrea, A. ; Song, I. ; Davis, K. C.: Analytics over large-scale multidimensional Data: The Big Data Revolution! In: *Proceedings of the ACM 14th international Workshop on Data Warehousing and OLAP* ACM, 2011, S. 101–104

[DG04] Dean, J. ; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. (2004), S. 137–150

[DJB19] D. Jackett, Tr. M. ; Barker, P.: Matlab Toolbox gsw_SA_from_SP_Baltic in TEOS-10. (2019). – www.teos-10.org

[DK13] Das, T.K. ; Kumar, P. M.: Big Data Analytics: A Framework for unstructured Data Analysis. In: *International Journal of Engineering Science & Technology* 5 (2013), Nr. 1, S. 153

[FC13] Fotache, M. ; Cogean, D.: NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL. In: *Informatica Economica* 17 (2013), Nr. 2

[FWW+10] Feistel, R. ; Weinreben, S. ; Wolf, H. ; Seitz, S. ; Spitzer, P. ; Adel, B. ; Nausch, G. ; Schneider, B. ; Wright, D.G.: Density and Absolute Salinity of the Baltic Sea 2006–2009. In: *Ocean Science* 6 (2010), Nr. 1

[HDW86] Hill, K. ; Dauphinee, T. ; Woods, D.: The Extension of the Practical Salinity Scale 1978 to Low Salinities. In: *IEEE Journal of Oceanic Engineering* 11 (1986), Nr. 1, S. 109–112

[HHLD11] Han, J. ; Haihong, E. ; Le, G. ; Du, J.: Survey on NoSQL Databases. In: *2011 6th international Conference on pervasive Computing and Applications* IEEE, 2011, S. 363–366

[HR83] Haerder, Th. ; Reuter, A.: Principles of transaction-oriented Database Recovery. In: *ACM computing surveys (CSUR)* 15 (1983), Nr. 4, S. 287–317

[IBM15] IBM: Database Db2 for i SQL Reference. (2015). – www.ibm.com

[ISL10] IOC ; SCOR ; LAPSO: The International thermodynamic equation of seatater - 2010: Calculation and Use of thermodynamic Properties. In: *Intergovernmental Oceanographic Commission, Manuals and Guides* (2010), Nr. 56, S. 1–196

[JYBC15] Jung, M. ; Youn, S. ; Bae, J. ; Choi, Y.: A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment. In: *2015 8th International Conference on Database Theory and Application (DTA)* IEEE, 2015, S. 14–17

[KR13] Kaur, K. ; Rani, R.: Modeling and Querying Data in NoSQL Databases. In: *2013 IEEE International Conference on Big Data* IEEE, 2013, S. 1–7

[LM10] Lakshman, A. ; Malik, Pr.: Cassandra: a decentralized structured Storage System. In: *ACM SIGOPS Operating Systems Review* 44 (2010), Nr. 2, S. 35–40

[LM13] Li, Y. ; Manoharan, S.: A Performance Comparison of SQL and NoSQL Databases. In: *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* IEEE, 2013, S. 15–19

[Los13] Loshin, D.: Big Data Analytics: From strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph. (2013)

[Mat19] MathWorks: Matlab R2019b Documentation, Primer. (2019). – www.mathworks.com

[MB19] McDougall, Tr. ; Barker, P.: Matlab Toolbox gsw_SP_from_C in TEOS-10. (2019). – www.teos-10.org

[Mey16] Meyer, Fr.: Temporale Aspekte und Provenance-Anfragen im Umfeld des Forschungsdatenmanagements. (2016). – Master Thesis, University of Rostock (DBIS)

[MH13] Moniruzzaman, A. B. M. ; Hossain, S. A.: NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. In: *CoRR* abs/1307.0191 (2013)

[MH17] Marten, D. ; Heuer, A.: Machine Learning on Large Databases: Transforming Hidden Markov Models to SQL Statements. In: *Open Journal of Databases (OJDB)* 4 (2017), Nr. 1, S. 22–42

[Möl16] Möller, M. L.: Aufbau einer Forschungsdatenverwaltung für chemische und physikalische In-Situ-Daten aus der Ostsee. (2016). – Bachelor Thesis, University of Rostock (DBIS)

[Mom01] Momjian, Br.: PostgreSQL: Introduction and Concepts. 192 (2001). – Published by Addison-Wesley New York

[MTSA19] Makris, A. ; Tserpes, K. ; Spiliopoulos, G. ; Anagnostopoulos, D.: Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal Data. In: *EDBT/ICDT Workshops*, 2019

[NPP13] Nayak, A. ; Poriya, A. ; Poojary, D.: Type of NOSQL Databases and its Comparison with Relational Databases. In: *International Journal of Applied Information Systems* 5 (2013), Nr. 4, S. 16–19

[OBLB15] Oussous, A. ; Benjelloun, F. ; Lahcen, A. A. ; Belfkih, S.: Comparison and Classification of NoSQL Databases for Big Data. In: *Proceedings of International Conference on Big Data, Cloud and Applications* Bd. 2, 2015

[Ord13] Ordonez, C.: Can we analyze Big Data inside a DBMS? In: *Proceedings of the sixteenth international Workshop on Data Warehousing and OLAP* ACM, 2013, S. 85–92

[Pri19] Prien, R. D.: Profilierende Verankerung in der Gotlandsee: Mit neuen Technologien den Datenmangel bekämpfen. (2019). – Presentation, Leibniz-Institute for Baltic Sea Research, Warnemünde

[SK11] Strauch, Chr. ; Kriha, W.: NoSQL Databases. In: *Lecture Notes, Stuttgart Media University* (2011)

[Str98] Strozzi, C.: NoSQL: A relational Database Management System. (1998)

[TB11] Tudorica, B. G. ; Bucur, Cr.: A Comparison between several NoSQL Databases with Comments and Notes. In: *2011 RoEduNet international Conference 10th Edition: Networking in Education and Research* IEEE, 2011, S. 1–5

[Tre14] Trelle, T.: MongoDB: Der praktische Einstieg. (2014). – Published by dpunkt.verlag

[VFKV16] Venkatraman, S. ; Fahd, K. ; Kaspi, S. ; Venkatraman, R.: SQL versus NoSQL Movement with Big Data Analytics. In: *IJCSIT* 8 (2016), S. 59–66

[VVM12] Van der Veen, J. S. ; Van der Waaij, Br. ; Meijer, R. J.: Sensor data Storage Performance: SQL or NoSQL, physical or virtual. In: *2012 IEEE fifth international conference on cloud computing* IEEE, 2012, S. 431–438

[Web10] Weber, S.: NoSQL Databases. (2010). – Technical Report, University of Applied Sciences, HTW Chur, Switzerland

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rostock, den 11.02.2020 _____