Kristianstad University
SE-291 88 Kristianstad
Sweden
+46 44 250 30 00
www.hkr.se

Independent project (degree project), 15 credits, for the Degree of Bachelor of Science (180 credits) with a major in Computer Science
Spring Semester 2021
Faculty of Natural Sciences

A Performance Comparison between Graph Databases

Degree Project about the comparison between Neo4j, GraphDB and OrientDB on different operations

Alm Robert, Imeri Lavdim

Author
Alm Robert, Imeri Lavdim

Title
A Performance Comparison between Graph Databases

Supervisor
Nazila Hasanzade

Examiner
Dawit Mengistu

Abstract
in this research we study what is the theoretical complexity of Neo4J, OrientDB and GraphDB, (three known Graph Databases that can be accessed by a Java instance), and how this complexity is manifested in a real life performance, To study their practical performance, a software was implemented and named as a profiler, which is capable to profile, (to record the time that is needed), each operation, and display the results in an accurate and organized manner. The technical documentation of those 3 databases was reviewed as well, to identify how the databases work, and what are their strong and weak points.
By the profiling process, the best performance was displayed by Neo4J, and while OrientDB failed to deliver, GraphDB takes the second place in terms of performance. We can identify a potential in OrientDB's approach, but its structure is too complex and rigid. Neo4J has a robust structure and an architecture that gives to it a great performance, while the Cypher syntax, which Neo4J uses, minimizes the possibility of human error. GraphDB is optimized for large scale public-data operations but performs well as a stand-alone solution as well.

Keywords
GraphDB, OrientDB, Neo4j, Graph Databases, Profiling, Performance, Databases

Contents

Acknowledgements

We wish to express our gratitude to our supervisor Nazila Hasanzade, our examiner Dr Dawit Mengistu, the head of our class Dr Kamilla Klonowska, and everyone from HKR that contributed to make this Degree Thesis possible, not only for their efforts to run this process, but also for the consultation and the support that we received during our Thesis Project.

Also, we wish to extend our gratitude to Dr. Jim Webber from Neo4J and Robert Alm's brother Dr. Klas Håkan Alm from University of Boras, for their knowledge and their assistance.

At last, we wish of course to thank our family, our friends, and the Open-Source Community for helping us to do what we love, namely programming.

Abbreviations

Table 1: Frequently Used Abbreviations

SQL	Structured Query Language
NoSQL	It does not use SQL
CRUD	Create, Read, Update, Delete
OOP	Object Oriented Programming
ACID	Atomicity, Consistency, Isolation, and Durability
BASE	Basic Availability, Soft-state and Eventual consistency
Neo4J	Network Exploration and Optimization "for" Java
SPARQL	SPARQL Protocol and RDF Query Language
RDF	Resource Description Framework
W3C	World Wide Web Consortium
HTTP	Hypertext Transfer Protocol
XML	Extensible Markup Language
RSS	Really Simple Syndication

Introduction

Problem definition

The aim is to compare different NoSQL graph databases in terms of complexity and performance. The Databases that will be investigated are Neo4j, GraphDB and OrientDB, which have been chosen because they fulfil the criteria of being Graph Databases and using Java. The project also aims to draw conclusions about in which case each of the Databases performs better and why.

Motivation

As the global computational needs are increasing, traditional Databases, for example Relational Databases, struggle to cope with the technological advancement of human society, and that highlights the need for new solutions that can improve, not only the speed, but also the robustness and the flexibility of a database. As it is stated in "A Comparison of Current Graph Database Models" by Renzo Angles [1]:

> *"The limitations of traditional databases, in particular the relational model, to cover the requirements of current applications has led the development of new database technologies. Among them, the Graph Databases are calling the attention of the database community because in trendy projects where a database is needed, the extraction of worthy information relies on processing the graph-like structure of the data."*

NoSQL Databases are competing against the traditional Databases for at least a decade now, and this continuing innovation brought forward a new kind of NoSQL Databases, namely the Graph Databases. Graph Databases are promising due to their ability to use Graph Algorithms to their advantage, and for their ability to make connections between their data, a thing that is, for example, extremely important for Artificial Intelligence.

Graph Databases are Databases that are based to Graph Structures, and likewise the typical Graph Data Structures, Graph Databases consist as well of Nodes, Edges and Attributes, (that store data inside the Nodes and the Edges). While Graph Databases are considered of type NoSQL, they have an element that makes them very different from the regular NoSQL Databases which is the existence of relations. Relations in Graph Databases are represented as Edges, and they stored and used as such. This characteristic makes them more efficient for relation-based queries and their graph structure gives the opportunity to Graph Databases to be traversed with the support of the existent Graph Algorithms, thing that makes them competitive against other types of databases, (especially the ones that are not designed to handle relations).

As different Graph Databases are getting gathered to compete with the other Databases and the other Graph Databases, it gets increasingly important to know which Graph Database are capable to utilize Graph Algorithms , and how we can use each of them in an optimal way.

The idea is about a qualitative comparison between Neo4j, GraphDB and OrientDB, (three graph databases that were selected because they are compatible with Java), with a final goal to draw a conclusion about which of those three Databases is suitable for each different CRUD, (Create Read, Update, Delete), operation. Research literature will be used in a large scale, as the best source of information now is the existent documentation for each database, and there is the desire to use profiling as well to compare, (in a smaller scale), the performance of each Database on CRUD operations for each of the selected databases. The specific three databases were selected because of their popularity and because they are compatible with Java which we intent to use for profiling.

The authors, (R. Alm and L. Imeri), wish to consider analysing each of the three Graph Databases with the goal to classify the different characteristics of each Database, and how each of them can be used properly.

Source of Inspiration for this Idea are the books, “Databases Illuminated” [2] and “Data Structures and Algorithmic Analysis in Java” [3], and it is expected that other publications about the topic will be used, as the complexity of the topic requires deep and variant research literature. The special technical characteristics of the many known Databases are highlighting the need for the use of a technical documentation for each of the selected databases, which can be found in the official web pages of each Database.

Research Questions

1. What is the theoretical complexity on CRUD operations of Neo4J, GraphDB and OrientDB Databases?

2. What is the practical performance of Neo4j, GraphDB and OrientDB on their CRUD operations in terms of time?

3. How do the theoretical complexity conforms with the practical performance of Neo4j, GraphDB and OrientDB. Why do those three Databases act that way?

Methodology

Literature Review (for the research questions 1 and 3)

The basic component to understand each Database it is its own documentation and the work that have been done before by others to categorize the various technical characteristics, so before any analysis or any profiling takes place, it is essential to review and to study the literature that exists about the performance, the complexity, and the technical characteristics of each database with focus to the documentation that is provided by the official sources of each Database.

Profiling, (for the research question 2)

To answer the second research question, experimental methodology is necessary. The method that will be followed will be an experimental method which is called profiling which is the process of counting the time which is necessary to complete a specific task.

For our experiment, the specific task will be the four operations of CRUD upon the least common unit of operation between the three Graph Databases which is a node. With consideration that all the Databases in question are Graph Databases, a node is selected as it is the very basic element of their data structure and it plays the very same role that an entry or a row, (for a relational database), for one other database.

As we wish to have an idea about the performance of each Database, profiling becomes necessary. To achieve an objective and well-constructed profiling, few rules will have to take place.

1. All three of the Databases will have to be profiled locally and in the same computer and in the same conditions.
2. All the tested Database will have to be accessed by their most recommended DBMS to exclude cases that are too simplistic or primitive, (for example the access through a simple file or variable).
3. All Databases that must be tested should be compatible with Java because we intend to use Java for profiling.

4. All the CRUD operations will be tested separately by having the node as a common unit of operation. CREATE will mean to create a node with a specific content, for each unit, READ will mean to read the content of a node for each unit, UPDATE will mean to edit a specific element inside a node for each unit and DELETE will mean to delete a node for each unit.
5. The units are used to increase or to decrease the scale of the experiment, and it indicates how many nodes will be used for each operation. For example, if the scale is 1000 units for the operation CREATE, 1000 nodes will be created with the same content, (their id of course will be unique and different), for each of the three Graph Databases. Same process of course will be followed for the other operations as well.
6. For all the operations the timer for the profiling will start exactly when the number of specific operations will be requested from a Java terminal and it will stop when the number of operations will be complete.
7. Same process will be followed for all of four operations and all three Graph Databases and for different sets of units but it has to be a different profiling session for each case. That means for four CRUD operations on three Graph Databases and, (for example), 5 different scales, (sets of units, multiplies of 10), 4x3x5=60 different profiling sessions will take place.

Analysis, (for the research questions 1, 2 and 3)

After completing the Literature Review and the Profiling, we can use the received data and to start drawing conclusions.

We will take the data which is fetched from the profiling, and we will identify where each Graph Database performed better or worse from the other databases and we will try to map the performance differences to the structure and the logical workflow of each Graph Database.

First, the logical workflow of each Database will be theoretically analysed to recognize its strong and week points. The very next step will be to compare the theoretical analysis with the profiling data to see if the profiling corresponds the theoretical benefits and drawbacks of each Database and why.

Expected Results

The expected data will be displayed, (in two-dimensional diagrams), indicating how well each Database performs in each different profiling session, and together with the analysis from the Literature Review will be able to highlight the strong and the week points in the workflow of each Graph Database and other possible strengths or limitations

Literature

In our efforts to collect the theory for our research we followed 3 ways.

First, we contacted the three companies, (Neo4J, Ontotext for GraphDB and SAP for OrientDB). While SAP and Ontotext weren't available to contact with us, we received a message from Dr. Jim Webber a Chief Scientist at Neo4j, that highlighted the strong points of Neo4J database, (and DBMS), namely the traversal performance, the ACID guaranties, and the humane query language of Neo4j, (Cypher). Also, two advices were given to us, first to use the recommended query language, (Cypher), to unlock the full potential of the database, and to perform a hostile testing approach, to highlight the level of competence of each DBMS to protect the data under pressure.

Another way to collect information from the source was by reading the documentation of the products that we refer in this research, (Neo4J, OrientDB, GraphDB. etc), as the documentation of the developers is the very first source of information about the developed product.

For the companies and the developing teams, we used their official pages and their "about us" sections to gather information, and for conventions and standards we used the pages of the official publications of each standard in the pages of the Organizations that published those standards. In case of algorithms that are referred here, we use either the

official publication, (if applicable), or a publication about the algorithm, preferably from a recognized practitioner.

Related Work

While having the collection of our knowledge collected directly from the documentations of the various products that we use, related work serves often as an inspiration and it can support a few of our statements if a further research is possible.

During our research we discuss the role that the RDF structure from the GraphDB database has to play for a large-scale applications, a discussion that jumps out of the scope of our research and it is left for future work. A related work that can serve as a guide and as an inspiration for this kind of research can be the publication from V. Kotsev, N. Minadakis, V. Papakonstantinou, O. Erling, I. Fundulaki, and A. Kiryakov, with the title "Benchmarking RDF Query Engines:

The LDBC Semantic Publishing Benchmark" [4] publication describes how SPARQL and RDF protocol can be used for meaningful use of enormous size of data, to enable querying and analysis on this data, and what are the challenges of this approach. The fact that the publication refers to GraphDB, makes it an important reference for any future study of GraphDB.

For Neo4J a literature review was published on 2016 with the title "Literature review about Neo4j graph database as a feasible alternative for replacing RDBMS" [5] by Felix Melchor Santos Lopez and Eulogio Santos De la Cruz, and serves as a comparison between Neo4J and RDBMS, (Relational Database Management Systems), without targeting a specific RDBMS but by analysing how a NoSQL DBMS can offer an alternative to the traditional Relational Databases by increasing the flexibility, the scalability and the performance, highlighting in that way Neo4j behind the label NoSQL.

OrientDB is mentioned among other databases on the publication with the title "Holistic evaluation in multi-model databases benchmarking" [6], which is focused mostly on the multi-model approach of OrientDB, while our research focuses mostly on its OOP characteristics. The publication discusses possibilities that can open through multi-

modelling and benchmarking and analyses how several databases, (including OrientDB) are using this approach. The publication published by Chao Zhang and Jiaheng Lu the December of 2019.

The publication that published in 2020 by Moath Jarrah, Bahaa Al-khatieb, Naseem Mahasneh, Baghdad Al-khateeb and Yaser Jararweh with the title "GDBApex: A graph-based system to enable efficient transformation of enterprise infrastructures" [7] gets deeper into the structure of OrientDB as it is using it as an example of a Graph Database and how it competes with relational databases. The publication focusses on the structure of both systems their layers and their different components and the design architecture that is used to improve the performance of a graph database. The publication also discusses about the different buffer layers that OrientDB uses to achieve a balanced load. We refer to that on later point when we discuss about the technical details of OrientDb and its OOP layer.

Neo4j

Neo4j, [8] (which is an abbreviation for Network Exploration and Optimization 4 Java), is a database software based on Java that serves as a Graph Database management system and it is described by its developers as an ACID [2] compliant database, (ACID is an abbreviation for Atomicity, Consistency, Isolation, Durability).

More specifically Neo4j is not just a single piece of software but a suite of different software that cover different needs. Understandably the Neo4j DBMS which is called "Neo4J Desktop" is the most representative software, but other pieces of software are a part of the neo4J platform, like Neo4j Browser which allows access and management of a Neo4J database through a typical web browser, minimizing that way the need for an installed software. In the same way, sandbox is accessible through a web browser, but it is used for testing as it holds any data only temporarily. Another important piece of software is an editor which is called Bloom, (Neo4j Bloom), and it is using advance

graphical illustrations, and tools to allow the user to manipulate and analyse the data in the database, without the need for code.

Understandably several other software that belong to the dynamic of the Neo4j platform, like connectors and libraries, with a characteristic example the "Neo4j Graph Data Science (GDS) Library" which is used widely for data analysis purposes.

Even though Neo4j is a Java based software, a declarative language is necessary for the needs of handling the data. That need is covered by the language which is called Cypher and it was implemented around 2011 by Neo4j company called "Neo Technology", (today the company is called Neo4j inc).

Neo4j started as an idea around 2000 when the founders of Neo4j encountered difficulties with the existing Database Management Systems, (for Relational Databases), and started working on a new DBMS software, namely Neo4j, which got implemented as a first version around 2002, and it got released as an Open Source Software under a GPL licence [9], by the Swedish, (back then), company.

The first version of Neo4j, (version 1.0), was released around 2010 and in 2011 the headquarters of Neo4j moved to Silicon Valley, (an area that is located in San Francisco Bay Area in Northern California, a location known for housing many larger Computer software and hardware companies).

Neo4j version 3.0 was released in 2016, followed one year later by the release of Neo4j Desktop, and the release of the Neo4j Bloom software around 2018.

Around 2019, Neo4j Released "Neo4j Aura", which can be described as a cloud platform for Neo4j Databases, a popular choice among Neo4j Developers due to its characteristics.

GraphDB by Ontotext

Ontotext [10] is a Bulgarian software company and its focus is the development

of a software, based on the Semantic Web languages and standards such as RDF, OWL

and SPARQL.

The popularity of the Ontotext software company comes from one of its products, that is called GraphDB, (previously known as BIGOWLIM) [11]. GraphDB capabilities like

text mining by using big knowledge graphs, are used for the development of an enterprise knowledge management and analytics system.

GraphDB is part of highly efficient, robust and scalable RDF [12] databases. To be easy to use and compatible with the industry, RDF4J framework interfaces, the W3C [13] SPARQL [14] Protocol specification, and all the RDF serialization formats are implemented and supported by GraphDB. It is a graph database which can be used for small projects or to be part of big organizations, and some of the reasons for that are "cluster support" and the capability of integration with external high-performance search applications such as Lucene, Solr and ElasticSearch.

An important feature of the GraphDB is its ability to allow users derive new semantic facts from existing facts by performing "semantic inferencing". Graph databases usually contain this feature, but what makes GraphDD special in this case, is the fact that it is triplestore and only few triplestores can perform this operation. The Reasoner, (TREEE Engine), is the one which performs the inference, where highly optimized data structures that kept in the memory, (for query evaluation and further inference), store explicit and inferred statements. In this research we tend to refer to the triadic character of the RDF storage, and by that we mean that the records are structured with links, (prefix, suffix, and predicate), and the reason why we don't refer to the triplestores is because we refer to the structure, (the "triadic" structure), and not to the protocol that utilizes it.

GraphDB is a part of RDF4J framework, packaged as a "Storage And Interface Layer", (SAIL) and makes large use of the features and the infrastructure of the framework.

RDF model, RDF parsers and query engines are the most important features used by GraphDB. To be integrated with the rest of the RDF4J framework such as query engines and the WEB UI, The Sail API interface is implemented by GraphDB.

GraphDB Workbench is a web-based administration tool that is used for administrative tasks. GraphDB workbench is easy, intuitive to use, and it has capabilities for managing, configuring and performance tuning "GraphDB repositories". GraphDB Workbench can be used to manage GraphDB repositories, loading and exporting data, monitoring query execution, developing, executing queries, managing connectors, and managing users.

OrientDB

OrientDB [15] which is released As an open source project, (Apache 2 License [16]), by OrientDB Ltd around 2010 is a diverse and very versatile Database Management System, (DBMS), based on Java, that includes a Graph Database functionality among other NoSQL [2] database models, with emphasis to a Graph and relation-oriented structure and functionality. Its special characteristic is that it generates a surrogate key, (a key that is generated with the purpose to be unique, and it is not being taken directly by a piece of data inside the database), that helps mapping the entries in a specific location inside an array List, offering that way, and easy and robust way to add and remove new entries, and also the possibility for fast search by using hash and graph algorithms.

The greatest achievement of OrientDB was around 2015 when OrientDB won the 2015 InfoWorld Bossie award. OrientDB Ltd was acquired by CallidusCloud on 2017 and one year later, (2018), by SAP [17], a German software corporation, and it needs to be highlighted that many parts of the official page of OrientDB, (https://www.orientdb.org), are inactive with a page-migration shield page which referring to an email for inquiries about "SAP Enterprise OrientDB". The documentation about the OrientDB DBMS is included and updated inside the official page of OrientDB and the software ist still updated and upgraded with the newest to this date stable release 3.1.9 being released on 24/02/2021. The 2 versions of OrientDB are the community edition and the enterprise edition.

How the Data is stored

An essential element to understand the theoretical complexity and the characteristic of a database is the how each database stores its data, as it defines how the data will be accessed or modified.

A major factor of complexity to a database can be the existence of schema, a structure that is usually found on Relational databases [18], (SQL), but in our case it exists on OrientDB while Neo4j and GRaphDB are considered Schema-less [19]. Schema is a strict structure that defines explicitly how the data will be stored and it is used to increase the control and the security to a database, but it increases the cost overhead as well as it more cost effective to store the data directly to a document instead of having to be filtered

through a schema first. It needs to be mentioned that it is difficult for databases to be completely schema-less and in many cases the functionality of schema is transferred to the application layer of the DBMS.

All three databases are considered as NoSQL, but OrientDB has a structure that is very close to SQL [20]. Neo4J and OrientDB belong to ACID consistency model, while GraphDB is considered to belong to the BASE consistency model. All the three Databases have some level of OOP, (Object Oriented Programming), layer, but it is OrientDB that utilizes fully this layer as GraphDB works through web protocols and Neo4j focus on writing to the documents as soon as possible to achieve the reliability of ACID, (OrientDB uses Schema to achieve that). Neo4j and GraphDB are using a structure of triads [21], (Subject, Predicate and Object, for GraphDB we use the term triadic to refer to the structure of its stored data that has three links), while OrientDB follows a more complicated approach due to its OOP nature.

Neo4J [19] stores its data in files with the extension ".db" and it uses files as major categories, ("nodestore.db" is used to save the nodes, "relationshipstore.db" to store the edges/relationships, etc), and because it is schemaless it avoids to do large changes to its documents and that makes it more efficient.

GraphDB [22] is a web-based database that it focused on high availability as it contents can be accessed by many locations through web protocols. It uses the RDF protocol, (Resource Description Framework) [21], a model of displaying data on the web which is defined by World Wide Web Consortium, (known as W3C) [21] [13], on 2004. As expected, it uses ".rdf" which are accessed by URI, (Uniform, Resource, Identifier).

OreintDB [22] stores its data in files with the extension ".pcl" after a complicated sequence of buffers that manipulate the data in a very low level, and the data is stored in the form of "clusters" that resemble the "tables" on SQL. The focus of their DBMS is to utilize the memory in a sequence of buffers for faster access to the data, with its mechanism ensuring the reliability of the data. Also, OrientDB has strict control over how the data will be stored to the disc and the memory, which is an approach that aims to better performance.

ACID and BASE consistency models

While, theoretically, many consistency models can exist, as they are conventions, and consistency can cover many different needs, at the moment, two are the presiding models that affect the structure of the known Databases and those models are namely ACID and BASE. ACID model which is preferred by both the SQL and the NoSQL,(Graph), Databases can be described as a model with a tight consistency, as the ACID Databases trade-off their performance for more integrity. On the other side, BASE model is a model of a loose consistency that trades off its integrity for a better performance and scalability, (and more availability).

ACID is an abbreviation for Atomic, Consistent, Isolated and Durable consistency. By Atomic we mean that, either all operations in a transaction succeed, or that all the operations in the transaction fail, and in this case, the system rolls back before the moment of the transaction. By consistent we mean, that a transaction is succeed, only if the Data in the Database is consistent and well-structured after the transaction. By isolated we mean, that the data of each transaction and each operation, remains encapsulated inside its transaction/encapsulation, without effecting other data outside its area of influence, and without being affected by any external data. At last, by durable we mean that the data, after a successful transaction must be written in the system permanently and unaffected from any system failures.

BASE is an abbreviation for Basic Availability, Soft-state and Eventual consistency. By basic availability, of course we mean that the system trades-off part of its integrity for a better performance, that leads to better availability. “Soft-state” moves to the same direction, as the need for consistent data is minimized, and small alterations between replicated data are allowed. Of course, consistency is always the goal of a Database, but with BASE the consistency is allowed to be achieved after longer time, and with bigger fault tolerance than ACID, and that is what we mean by the term "Eventual Consistency", which is the last letter of the BASE abbreviation.

Profiling Experiment

To answer the second research question, ("What is the practical performance of Neo4j, GraphDB and OrientDB on their CRUD operations in terms of time?"), experimental methodology is necessary. The method that is followed is an experimental method which is called profiling which is the process of counting the time which is necessary to complete a specific task. For our experiment, the specific task needs to be the operations of CRUD upon the least common unit of operation between the three Graph Databases which is a node. With consideration that all the Databases in question are Graph Databases, a node is selected as it is the very basic element of their data structure and it plays the very same role that an entry or a row, (for a relational database), for one other database.

Strategy of the Implementation

The program designed to perform the same script multiple times, (rounds), and to deliver the results in an organized manner when all the rounds are done. Also, the script is designed to profile up to 6 different scales in the same round, (10, 100, 1000, 10k, 100k and 1M), and the different results from the timed operations were stored in different arrays with the same name with the scales, and with the same position as the number or the round, (the rounds start counting from zero and end up to the selected number of repetitions minus one).

The timer is designed to start and stop in such way that is does not get affected by other operations than the CRUD operations and includes the time to establish, (and eventually to drop), a connection with the database.

In order to avoid a situation where the system sends the query for the operation and proceeds, which it would not be optimal for the profiling, we designed the whole system to not proceed if no returning value is received, and in the lowest level of operation, return is achieved by returning the results of the query, which most of the times is accompanied by a select operation.

We used three extra classes, (Neo4jcrud, Orientdbcrud and Graphdbcrud), as an interface to the classes that we are using to access the databases, (Neo4jDB, OrientDbDB and GraphDbDB), for reasons of consistency and convenience.

We also used a unified message content for all the operations, which is based in the current date, the current round, the scale, the number of the operation and we included a special keyword in string format that we call a "custom seed", and it is used in order to achieve debugging when it is necessary.

The use of a unified message came from the decision to leave the identities of the nodes as they are and select the nodes through their messages. that way, the system knows always what node to create, delete, or update each time.

Installation of the databases

In order to let the profiler to connect to the databases. It was necessary to install a DBMS for each of the tested databases. The databases got installed locally in the same computer as the profiler software and they were configured in such way that they could work all of them at the same time if that was necessary, a feature that was not used due to the unnecessary complication that could bring, (each working DBMS can steal resources from the other DBMS and that could harm the accuracy of the system).

Installation of Neo4j Desktop

As it is explained in the official documentation of Neo4J [23], the installation of the Neo4j DBMS, namely “Neo4j Desktop” was an easy process as we only had to download and to run the installer for Neo4j Desktop version 1.4.7.

Creating and setting up the database was as well a simple and intuitive process, as we didn't have to run any command on the windows command prompt nor to run a batch script other than the executable of the installer. An impressive feature is that we were able to select which version of neo4j we wanted, a feature that usually is not offered by other DBMS.

Installation of GraphDB

In our attempt to install a GraphDB DBMS for our needs we faced a conflict between the versions 9.8.0 and 9.8.1. The problem lies in the outdated documentation and the major difference between 9.8.0 and 9.8.1, namely that 9.8.0 as it is stated to the documentation [24] requires a licence key but 9.8.1 does not. We tried to install 9.8.0, (we were not informed about 9.8.1), and while the downloading process was successful, the licence key never delivered, and the support email was not active. In our efforts to find a solution we stumbled upon 9.8.1 version who had its own installer, and a licenced key was not required.

It should be mentioned that while 9.8.0 is an outdated web-browser interface, 9.8.1 is a robust standalone software, that worked flawlessly during the testing. We assume that we coincidently faced the transition between 9.8.0 and 9.8.1 and that it is not the company's fault that we had troubles with the installation. Even so, it would be easier for us to find our way around if we had technical support from the company. The installation of 9.8.1 got completed through an installer that got activated through an executable file.

Installation of OrientDB

The installation of OrientDb, (we installed the latest stable version – "OrientDB 3.2.0 GA Community Edition"), was relatively easy, despite that it has not an installer and we had to run a Batch script to install it. The server.bat file that needs to be run it can be found in the bin folder of the archived files folder that we downloaded from the official page [25].

The best source for information about the installation can be found in the readme file inside the root folder of the archive. No special actions or commands were necessary for the installation and the interface is a web interface that can be found in the localhost if the correct port is used (http://localhost:2480). We noticed after a few efforts to use it that the DBMS, (OrientDB 3.2.0 GA Community Edition), does not work as it should.

Technical issues with OrientDB

We installed and tried to use “OrientDB 3.2.0 GA Community Edition” which is practically the only version that we could use, as we didn't intend to pay for the enterprise edition, and it seems that OrientDB is in a period of inactivity and the acquisition of the enterprise edition may not be applicable.

So long the company is in a phase of transitioning to the new owner, (SAP [17]), only the documentation and the community part of their official page is active while all the other content leads to a shield screen of their new owner. No warning was provided for any possible technical difficulties, and we didn't receive any reply when we contacted the company.

We used the archived file for the installation of the community edition, and the installation went seemingly without a problem. We experienced problems after the installation with the database failing to deliver sporadically several unconnected with each other operations like the "log in" or deleting something from the database. Common error which it was produced both during the run of the profiler and when we performed operation through the database's web client was the error "Cannot update record the resource has restricted access" [26], while the system of allocating privileges to the users was very unresponsive. We consulted the documentation that refers to security constraints [27], but as we already made sure to give full privileges to our users the recommended solutions on the documentation were not applicable.

We were able to perform "Create" and "Update" operations, but the system was unable to deliver a "Delete" operation, and we decided to perform a deletion by using a technique called "delete by tombstone" which means that when we wanted to delete a node, we declared it “deleted”.

During the profiling the database failed to deliver the necessary operations and for that reason it was marked as "Failed to deliver" and we didn't compare it with the other two databases, (GraphDB and Neo4j).

The Profiler

In order to be able to access the three DBMS in an organized manner we had to build a program that runs a script with the different CRUD operations to the different databases and it profiles, (in terms of time), and it analyses the results, in a repetitive way that ensures accuracy.

The program is called a profiler and it is built in a Maven instance for Java 11. To keep the instance pure, we wanted to avoid using Maven but it is proven to be the simplest alternative, so we choose Maven to ensure the replication of the experiment as maven dependencies can be found in our own pom.xml file, while modules for direct use are difficult or even impossible to find. Also, we managed to use the three different maven dependencies for the three different databases in a well constructed way, thing that is kind of difficult to be replicated from scratch.

No package structure was used and we used the default package, as we kept the hierarchy and the structure of the classes files flat for reasons of convenience. The main class where the "main()" method is located was named “Main”, and we also used a Model class that we call "Dataholder" in order to store global variables mostly for reasons of configuration. In order to be able to calculate the time of each operation one more class was added to the structure, namely the "Timer" class.

For a smooth access and operation of the three CRUD operations 2 classes were used for each database, the one with the name "Crud" in order to invoke a CRUD operation from the Database, and a "DB" class which is used to actually connect to the database and to perform the desired operations. The "Crud" classes are "Neo4jcrud", "Orientdbcrud" and "Graphdbcrud" while the "DB" classes are "Neo4jDB", "OrientDbDB" and "GraphDbDB"

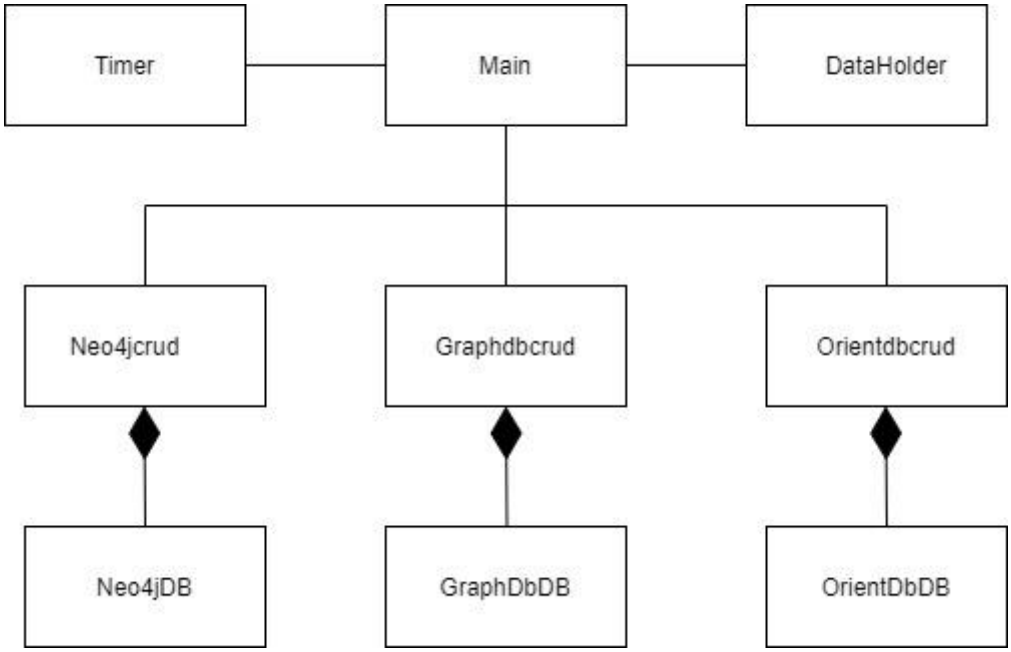

Figure 1: UML Class Diagram of the Profiler

Main

Inside the "main()" method in the class "Main" four important methods are called, the "menu()", the "run()", the "displayer()" and of course the “analyzer()”. The "menu()" asks the user about how many repetitions, (repetitions or samples are used for accuracy), they wish to have, which database to test, and for which CRUD operation, (Create, Update, Delete). The "run()" method runs the scripts according to preferences of the user. The "displayer()" displays the results of the profiling in an organized manner, and finally the "analyzer()" concludes a simple algorithmic analysis with the aim to support any useful conclusions.

Inside the "run()" and for each database there are 2 nested scripts. For example, for Neo4j, the method "scriptNeo4j()" makes sure to invoke the necessary script for each interval of operations that needs to be tested, and it makes sures to store the results in their specific location. It does that by calling the "scriptNeo4jNtimes()" by giving the operation and what size of interval to run. and used the specific array for its interval to store the result according to which repetition it is, (which round it is).

The stored results are used by the "displayer()" and the "analyzer()" to process the stored data and to deliver the necessary profiling results.

Dataholder

Dataholder is a Model Class and a convention that contains global variables, mostly for configuration's purposes. The majority of the global variables inside the class are boolean variables and they are used to configurate the system. The configurations are about if a connection to databases is allowed, in which databases the connection is allowed, how large intervals of operations are allowed to be used, and of course, the allowance of the option to print statements for implementations and testing. During the profiling the use of printed statements is not desired, because printed statements tend to be costly on resources and they affect the accuracy of the profiling. Also a few global variables inside the Dataholder are dedicated to the generation of a seed, which is an autogenerated String message that is used in the operations in an unified way.

The idea behind the seed is to use an unified message that applies as a payload to all the operations, and it is modified according to its interval, its round and other variables of the system, in order for the affected nodes to be detectable when it is necessary, (the delete operation needs to know what to delete, same goes to the update operation).

Timer

Timer is the class that works as a blueprint for objects that count and return the calculated time. it is used for the profiling the used operations.

Neo4jcrud, Orientdbcrud and Graphdbcrud

These three classes are working as an interface, (not a Java interface), between the "Main" class and the classes that handle the connection and the operations of the database. The basic methods are the "create()", the "update()" and the "delete()". Depending on the structure and the characteristics of each database, additional methods can be found in order to assure a better control of the flow of the operations.

Neo4jDB, OrientDbDB and GraphDbDB

These three classes are responsible for the connection to the database and for performing the actual operations. They are designed according to the recommendations of each database's documentation and they used just to perform a CRUD and they don't have more functionality apart of that as any extra functionality is usually performed on the crud classes or the "Main".

The profiling process

When the profiler was completed, and all the databases had a successful handshake with the profiler and a few test operations through the profiler it was decided that it was time to start the profiling.

The first database to be profiled was Neo4j, (we will call it Neo4j(1) for reasons of convenience), as we were confident about its performance and we aimed to know how far the profiler can go. After an unsuccessful try to reach the interval of 1 million on Create Operations on Neo4j, it was decided to now proceed with 1 million intervals.

The second try was 5 rounds of profiling with up to 100 thousand intervals. The profiling session was concluded after a half hour.

We tried to do the same with Delete and eventually with Update sessions but the operations, (especially on Delete), were taking too much time. We ended up profiling the operations Delete and Update on Neo4j for 2 rounds and up to 1000s intervals. We decided to keep our intervals up to 1000, deactivating that way the 10K, the 100K and the 1 million intervals.

After Neo4j(1) profiling was done, we profiled GraphDB in three separated operations, (Create, Update, Delete), and for 2 repetitions of intervals that reach up to 1000 operations each. The process was flawless but surprisingly fast, (Neo4j was expected to have a better performance, but instead it was slower).

After concluding the profiling of GraphDB, we followed the same path and we tried to profile OrientDB database on the 3 CRUD operations, (Create, Update, Delete), and for 2 rounds. The database instead of concluding 6x3330=19980 operations in total, it did 14, and because we were aware that OrientDB database has technical problems we marked its performance as "Failed to deliver" and we proceeded.

Our hypothesis about Neo4j's slow performance was that the database had too many records, (3 millions records were stored in the database), and to test our hypothesis, we decided to perform a new profiling session of Neo4j, (under the marking Neo4j(2)), by profiling the 3 operations, (Create, Update, Delete), and for 2 repetitions of intervals up

to 1000 operations for each interval, by using this time an empty database. The profiling session was successful, and our hypothesis was confirmed.

Due to the special characteristics of the payload of each node and to achieve a flawless performance that leaves the databases empty at the end, for all the profiling sessions, apart of the Neo4j(1), we followed the following process

1)Create Operation, up to 1000's intervals and for 2 rounds, (recorded results)

2)Update Operation, up to 1000's intervals and for 2 rounds, (recorded results)

3)Create Operation, up to 1000's intervals and for 2 rounds

4)Delete Operation, up to 1000's intervals and for 2 rounds, (recorded results)

Results

Create on Neo4J (1)

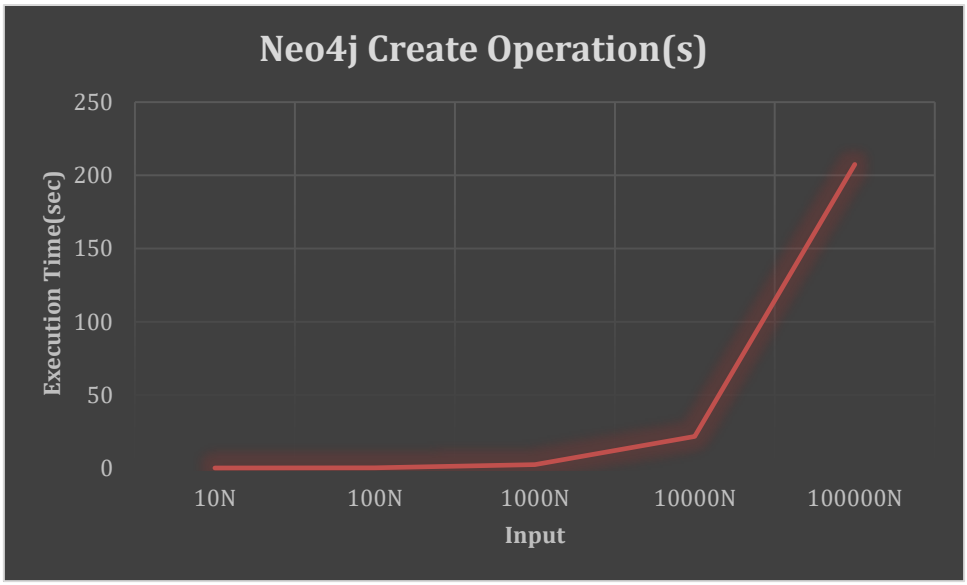

Figure 2: Profiling CREATE Operations on Neo4J

Table 2: THe value of profiling CREATE operations on Neo4J

10N	100N	1000N	10000N	100000N
0.23261	0.338612	2.613351	21.67792	207.4636

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=0.338611640/0.232609980=1

Analysis: <10, which indicates O(N)

N1000/N100=2.613350620/0.338611640=7

Analysis: <10, which indicates O(N)

N10000/N1000=21.677916600/2.613350620=8

Analysis: <10, which indicates O(N)

N100000/N10000=207.463572440/21.677916600=9

Analysis: <10, which indicates O(N)

Update on Neo4J (1)

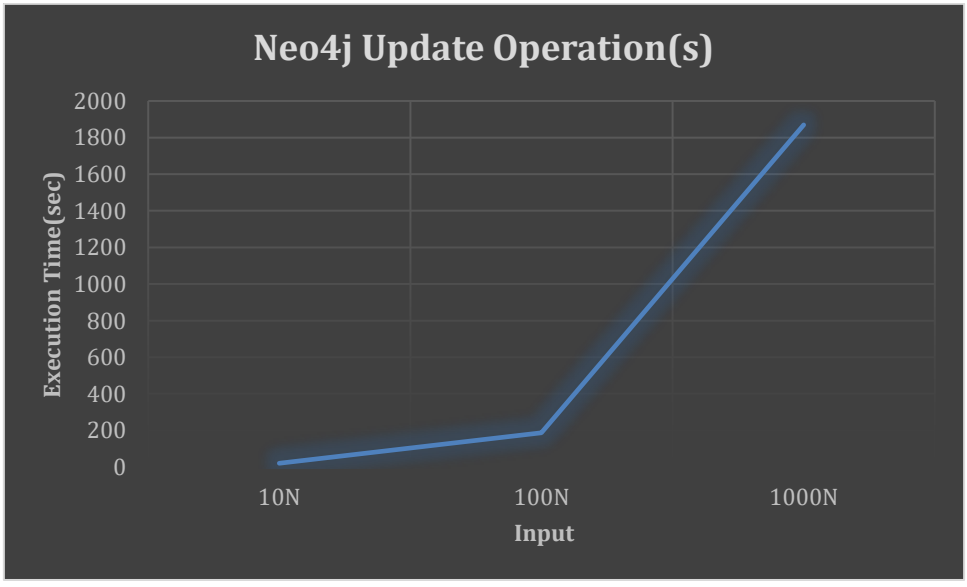

Figure 3: Profiling UPDATE operations on Neo4J

Table 3: Values from profiling UPDATE operations on Neo4J

10N	100N	1000N

21.162320750	188.208771550	1869.216492450

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=188.208771550/21.162320750=8

Analysis: <10, which indicates O(N)

N1000/N100=1869.216492450/188.208771550=9

Analysis: <10, which indicates O(N)

Delete on Neo4J (1)

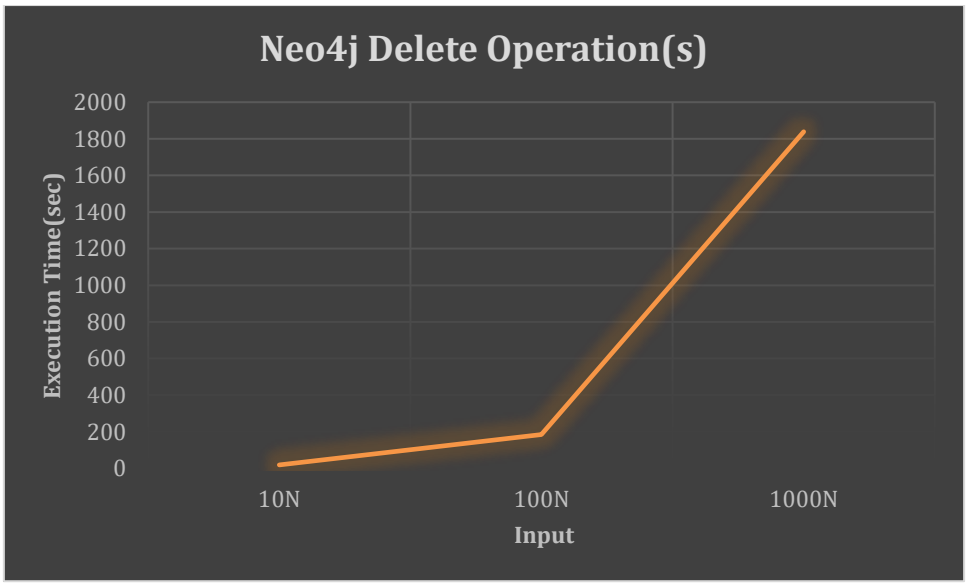

Figure 4: Profiling DELETE operations on Neo4J

Table 4: Values from profiling DELETE operation on Neo4J

10N	100N	1000N
18.934452350	184.252985000	1838.542065950

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=184.252985000/18.934452350=9

Analysis: <10, which indicates O(N)

N1000/N100=1838.542065950/184.252985000=9

Analysis: <10, which indicates O(N)

Create on GraphDB

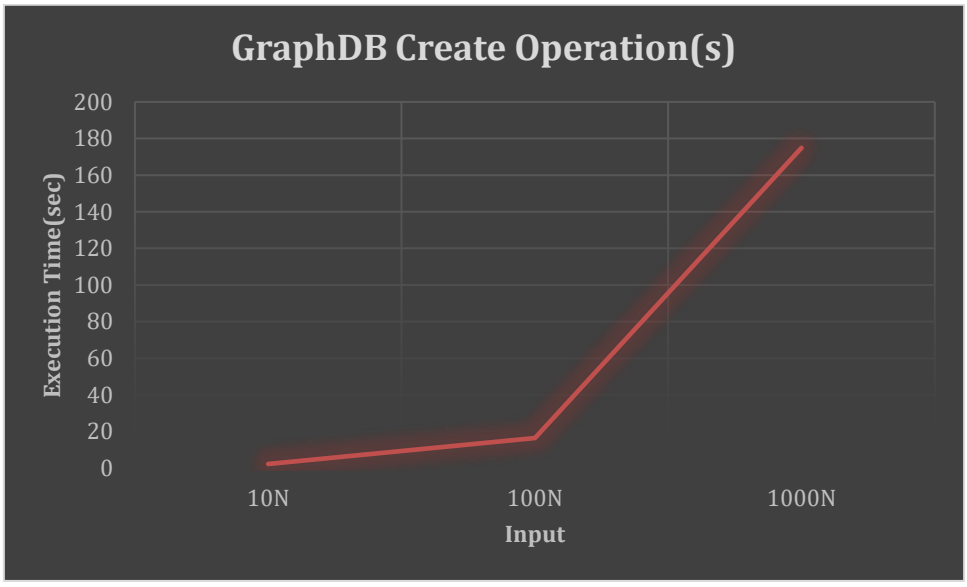

Figure 5: Profiling CREATE operations on GraphDB

Table 5: The values from profiling CREATE operation on GRaphDB

10N	100N	1000N
2.237987566	16.373772100	174.896962200

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=16.373772100/2.237987566=7

Analysis: <10, which indicates O(N)

N1000/N100=174.896962200/16.373772100=10

Analysis: equals 10, which indicates O(N)

Update on GraphDB

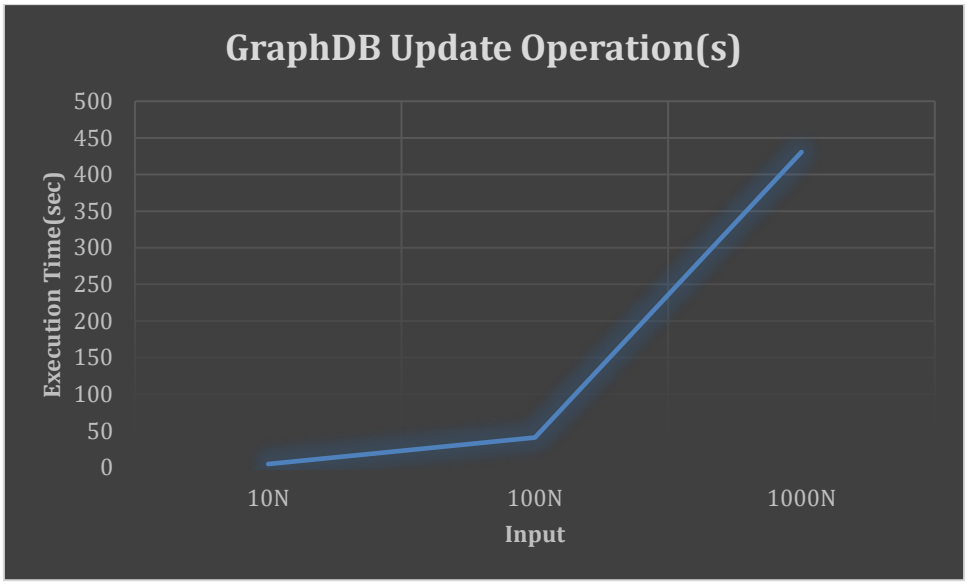

Figure 6: Profiling UPDATE operation on GraphDB

Table 6: The values from profiling UPDATE operation on GraphDB

10N	100N	1000N
4.656497566	40.846967033	430.648021866

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=40.846967033/4.656497566=8

Analysis: <10, which indicates O(N)

N1000/N100=430.648021866/40.846967033=10

Analysis: equals 10, which indicates O(N)

Delete on GraphDB

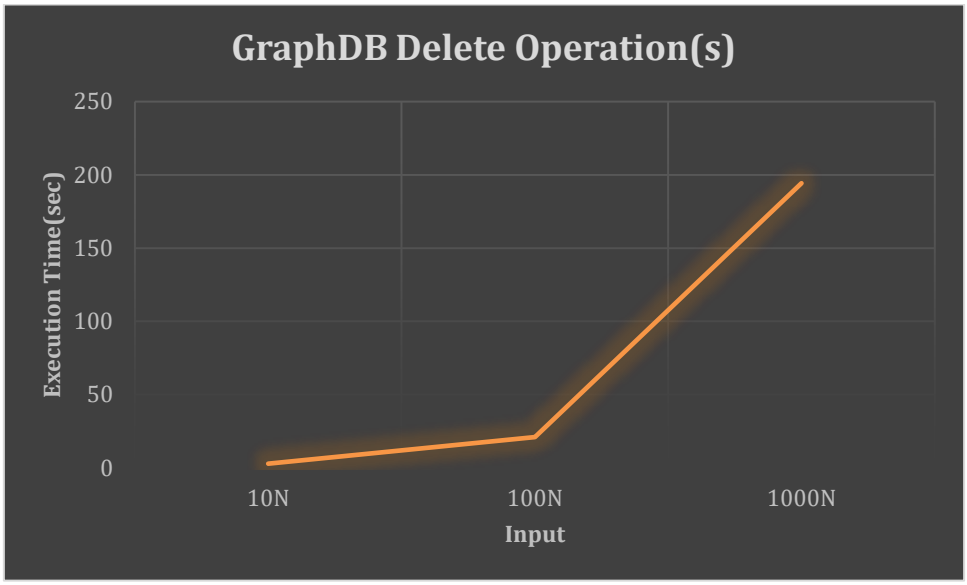

Figure 7: Profiling DELETE operations on GraphDB

Table 7: The values from profiling DELETE operations on GraphDB

10N	100N	1000N
2.822212766	20.908298766	194.275061200

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=20.908298766/2.822212766=7

Analysis: <10, which indicates O(N)

N1000/N100=194.275061200/20.908298766=9

Analysis: <10, which indicates O(N)

Create on Neo4J (2)

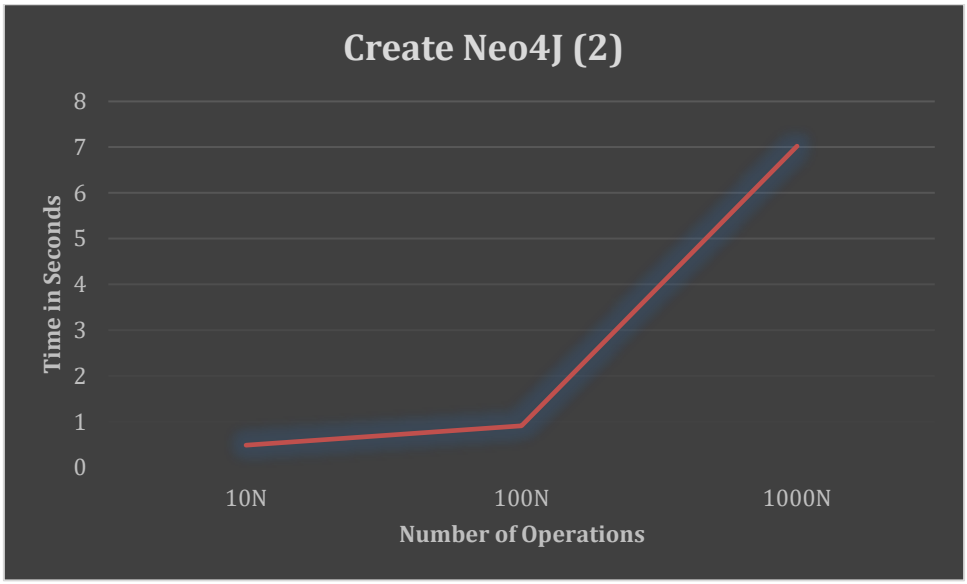

Figure 8: Profiling CREATE operations on Neo4J for second time with an empty database

Table 8: The values from the second profiling of CREATE operations on Neo4J

10N	100N	1000N
0.489384333	0.913828966	7.022316800

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=0.913828966/0.489384333=1

Analysis: <10, which indicates O(N)

N1000/N100=7.022316800/0.913828966=7

Analysis: <10, which indicates O(N)

Update on Neo4J (2)

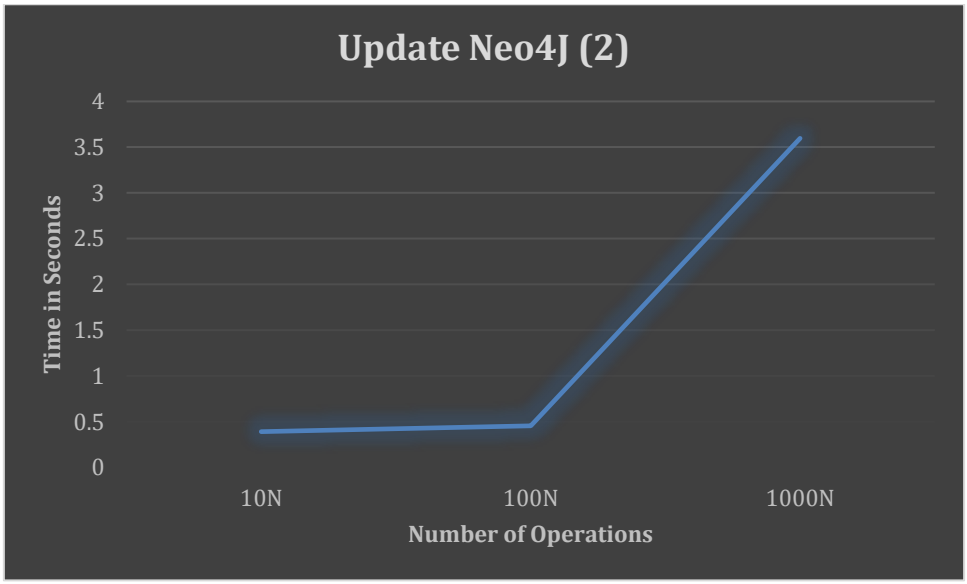

Figure 9: Profiling UPDATE operations on Neo4J for second time with an empty datavase

Table 9: Values from the second profiling of UPDATE operations on Neo4J

10N	100N	1000N
0.392272900	0.456272133	3.596461733

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=0.456272133/0.392272900=1

Analysis: <10, which indicates O(N)

N1000/N100=3.596461733/0.456272133=7

Analysis: <10, which indicates O(N)

Delete on Neo4J (2)

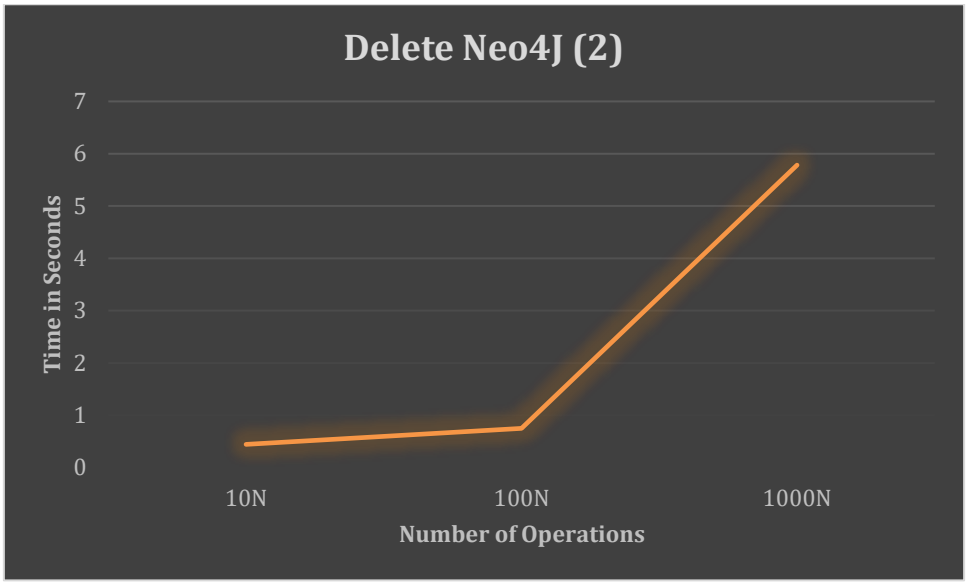

Figure 10: Profiling DELETE operations on Neo4J for second time with an empty database

Table 10: Values from the second profiling of DELETE operations on Neo4J

10N	100N	1000N
0.443387533	0.747109166	5.781791900

With increasing the input for 10 times the profiler gives the analysis in the form of ratio

N100/N10=0.747109166/0.443387533=1

Analysis: <10, which indicates O(N)

N1000/N100=5.781791900/0.747109166=7

Analysis: <10, which indicates O(N)

Discussion

Both OrientDB and GraphDB are showing signs of slowing down in their development, so it was easy to understand that Neo4j is going to give the best performance, but the

focus of this research is to find why we had this result and why each database/DBMS performs in that way.

First, OrientDB got excluded because it was malfunctioning. Our interpretation of the reason why OrientDB does not work is that because due to its unnecessary complexity and its change of ownership, without of course excluding the possibility of financial difficulties or other technical issues.

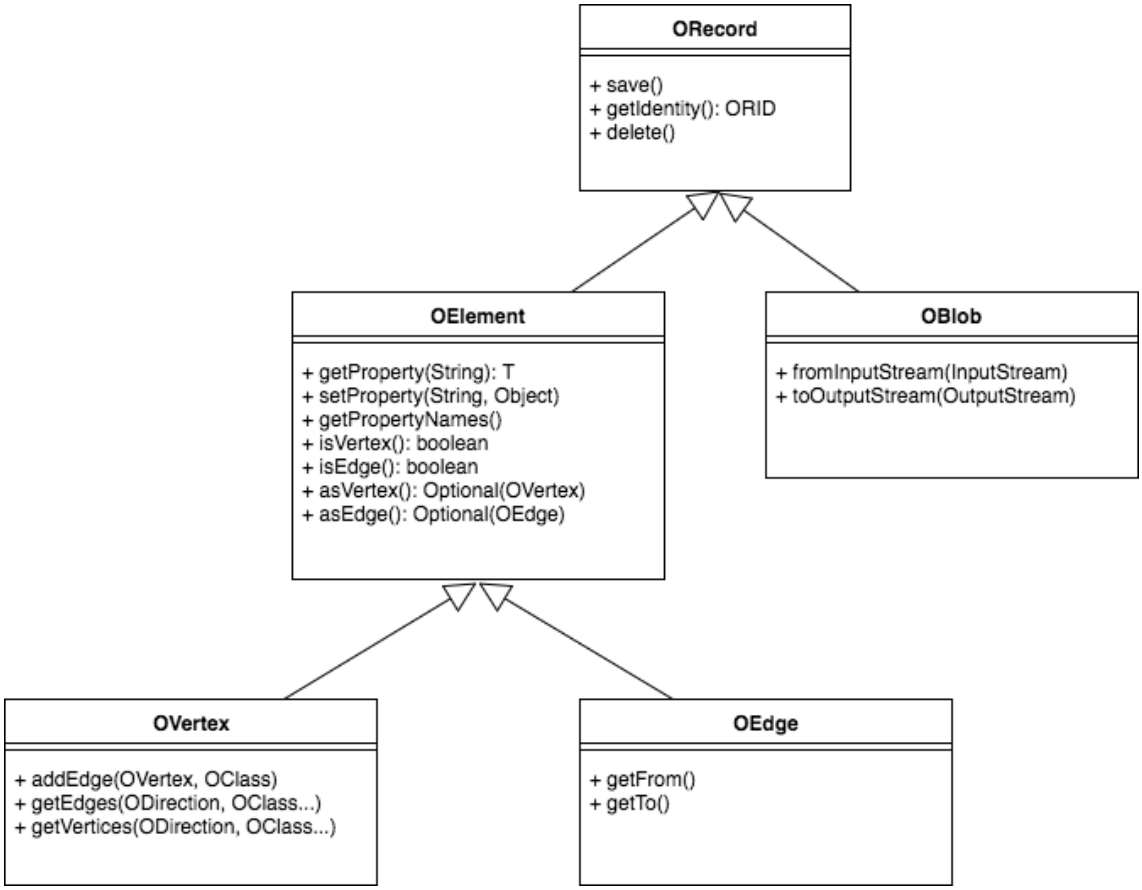

Figure 11: UML diagram that highlights the complexity that the users of OrientDB are facing

For a graph structure which means a node entity and a relationship entity, OrientDB offers to its users more than 5 classes to work with, (also the peripheral classes that exist to handle other functionalities of the client), and it is also implemented by using structures like "Java Interfaces" that are increasing the complexity even more. it is easy to understand that if such great number of classes are dedicated to being used from the client side, it can be only expected to have exponential increase of the complexity on the server side.

OrientDB is an OOP DBMS which has as its base a relational database model. That is great in terms of performance if it is implemented correctly, but it makes the code tedious to be implemented. The translation back and forth between the OOP layer and the data files is a process that need a good sync and good troubleshooting and identification mechanisms to make the system work. We detected that the system was "forgetting" itself and resetting, and that makes us believe that it is one or many of the sync and identification mechanisms that were broken and that is the reason why the system forgets its progress.

This issue could be caused because of untreated updates on a third party dependency, (the use of Java Interfaces can indicate the use of third party dependencies), but by taking to a consideration the change of ownership and the long period of inactivity, it might be an indication that a new developer team can't develop the code of the old one, (databases can be a very specific field of programming and the code usually tends to become tedious).

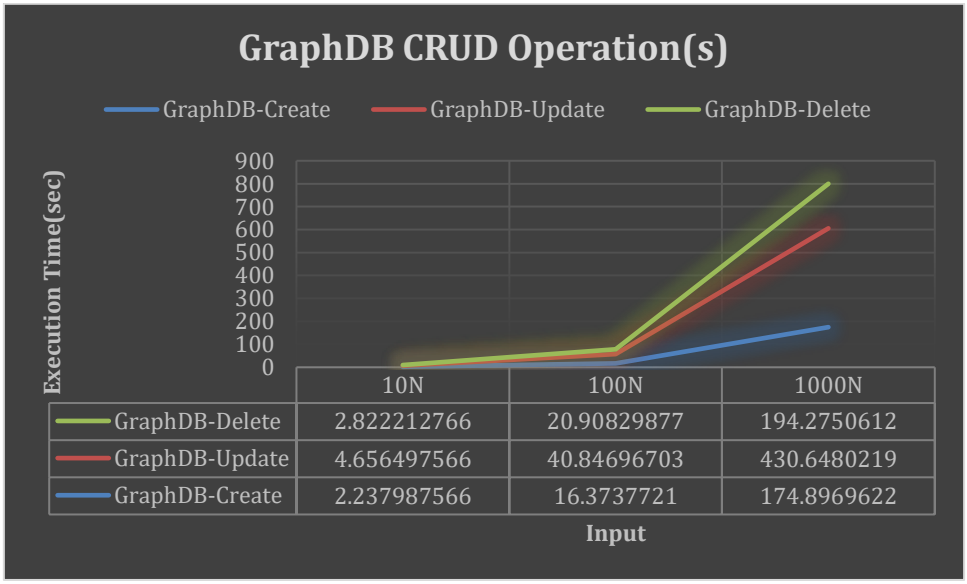

Figure 12: Full CRUD display of GRaphDB operations

In the comparison of performance between GraphDB and Neo4j, Neo4J performed better but that wasn't because of a malfunction or low quality of code from GraphDB's side, but mostly because Neo4J is optimized to be a standalone server, (with reservation that Neo4j has an enormous scalability), while GraphDB has another strategy, namely it uses the backbone of Web protocols like HTTP to perform its operations.

GraphDB uses Web records that are meant to be easily accessible to be read, but they need some security process in order to be updated. Theoretically security measures can be used to the READ operation as well, but that it is not an optimized option for GraphDB. The Web characteristics of GraphDB making it suitable for cases when a few users are updating the data and many users are reading the data. That can lead to a topology where many clients can use one, (or many - theoretically), server to instantly receive any published-on web records that are updated only by specific sources. A one-to-many or many-to-many topology that can perform in scale and to be used for broadcasting or in cases of display publicly available data. The advantage of GraphDB against other systems that display the same functionality it is the triadic structure on its records that allows the uses to do complicated select queries that makes it suitable for distributed scientific applications.

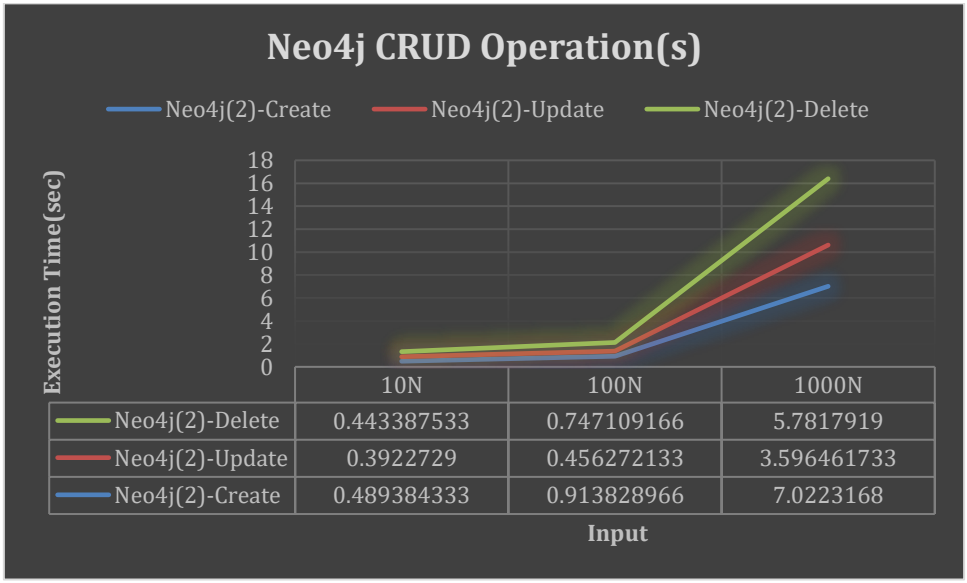

Figure 13: Full CRUD display from the first profiling on Neo4J. The first profiling got affected by the database that had stored 3 million records

When it comes to Neo4J, our decision to perform only single operations, instead of batch, prevented Neo4J to reach its full potential, something that was manifested when we accidentally performed operations in a database with 3 million records stored inside, and even if the performance of Neo4j was impressive, it seems that Neo4J's real performance is multiple times more efficient.

While the CREATE operation was the most accurate in terms of speed, the UPDATE and DELETE operations were missing the factor that Neo4j uses specific files for nodes and

relationships that forces a sequent iteration. Having to iterate all the similar components in same file, seems like an unproductive thing to do, but not if all the queries are satisfied in the same operation a thing that turns the complexity, (theoretically), of the SELECT operation to O(N) from O(N^2).

At this point we need to mention that any discussion about complexity remains hidden behind a general O(N) complexity which is happening because of the large time overhead that is allocated to each operation, making in that way all the operations to look like they have a linear complexity. The ratio of the algorithmic analysis remains applicable with the out(1000)/out(100) being the most important one as it escapes the overhead cost and displays an accurate ratio, with Neo4j having a ratio of 7 and GraphDB having a ratio between 8 and 10, that indicates that both Databases can perform better in a larger scale with Neo4J being able to perform better. The results of the profiler about the complexity of the CRUD operations, may require a further investigation before an O(N) average complexity can be confirmed.

We noticed that none of the Databases gave us the option to perform a Push/Pull or a Enqueue/Dequeue operation and while it is understandable why it is happening, it opens the discussion if there is something that needs to be improved. There are many applications that can make a use of Push/Pull or a Enqueue/Dequeue functionality, especially on pc-games, scientific applications, or servers, so there is some usefulness.

The lack of this feature seems to be because of the way that the data is stored. Simple key/value structures that are stored in form document are lacking the benefits of a tree data structure and that is the reason why they can't keep a track of the order of insertions, and they can not deliver any related operation. OrientDB could theoretically help the user to implement a way for performing a Push/Pull or an Enqueue/Dequeue operations, but we were not able to research it.

With the reservation that many graph algorithms, (like Union-find or Prim's algorithm), are still applicable to the tested graph databases due to their structure, it could be fair to state that the Graph characteristics of the databases are mostly a product of an embedded

normalization to their architecture instead of a Tree-like data structure. That is not a bad thing of course as the normalization in the architecture level can lead to a more flexible and more efficient database.

Our opinion is that such operations are still possible and that is because of the tendency of the tested DBMS to offer indexes in their features which can be used for more efficient searches. With the idea that everything can be an array in some way, it is maybe possible to use parts of the documents like the bytes, the characters, or the lines as pointers and to implement a system that track the order of the insertions due to their value, their name and their chronological order, thing that may increase the use of memory, but it can decrease the time of search and accessing each record. By following the example of the "Internet" it is possible to use Prim's algorithms or Dijkstra's algorithm to automatically map the internal structure of a graph data, in regular time intervals, and after that to use sorting algorithms to sort the records according to the needs of each database.

At last, and with the thought that databases should inherit characteristics from their most related data structures, we believe that there are many ways to make databases algorithm-friendly and to improving their performance even more.

Conclusion

Research Questions

Question 1: What is the theoretical complexity on CRUD operations of Neo4j, GraphDB and OrientDB Databases?

Answer: While OrientDB is excluded as its theoretical complexity can’t be compared with its experimental one, (because OrientDB does not work), all of the databases tend to have a theoretical complexity slightly below O(N) and with worst case O(N^2), which was not reached during our profiling, (apart of OrientDB which was excluded). The theoretical complexity was assumed, based on indicators and there is the opportunity for further study in future works.

Question 2: What is the practical performance of Neo4j, GraphDB and OrientDB on their CRUD operations in terms of time?

Answer: OrientDB Failed to deliver. The other 2 Databases had a standard duration for each operation which was a constant. Neo4J needed less time to perform each operation during the profiling in comparison with GraphDB's performance. The analysis of the performance indicates a linear regression.

Question 3: How do the theoretical complexity conforms with the practical performance of Neo4j, GraphDB and OrientDB. Why do those three Databases act that way?

Answer: Theoretically OrientDB uses the memory of the computer, which makes its operations to take less time to be performed, with the trade-off that a serious buffer management is needed. In practice OrientDB Failed to deliver, so, any benefit of its structure remains theoretical at the time. Neo4J which had the best performance, is optimized to be a stand-alone solution, but its file structure offers an impressive scalability as it allows many operations to be performed in each iteration.

GraphDB is designed to work on the backbone of the web, and it is a good solution for distributed systems, as it offers, theoretically, unlimited capacity and the ability to perform complex queries on a large amount of publicly available data.

Based on the our theoretical and practical indicators we can assume a O(N) average for at least the two of the Graph Databases, (Neo4J and GraphDB), but this can be a subject for further study.

Future Work

First, during the research we got a series of indicators that the theorecal and practical complexity of at least the two of the Graph Databases, (Neo4J and GraphDB), and that can lead to further research that confirms the results.

With OrientDB in an inactive state and Neo4j taking the lead in terms of advancement and performance, it is possible to highlight a few steps that can be taken in academic level to understand or even making improvements for both Neo4j and GraphDB.

GraphDB has a novel structure which can be used for a large BASE data-model while Neo4j is one of the top solutions for ACID data, thing that opens a discussion about a comparison between the 2 major ACID DBMS, namely MySQL and Neo4j.

A comparison of performance between Neo4j and MySQL is a highly desired analysis as it can point out which of those 2 DBMS can be used and were. Also, a market niche is needed to be found for systems like GraphDB, and any research that can help BASE databases to improve their scalability can be beneficial for the industry and the academic society.

A serious matter that got our attention is that Databases are not willing to inherit characteristics from the most similar data structures to them, (for example graph databases have many similarities with tree data structures etc).

The lack of inheritance was pointed out after our attempt to perform a "pop" operation for our Delete operations. The fact that if a graph database had a way to perform a pop, it would perform rapid Delete operations.

It is understandable that LIFO and FIFO features like push/pop and queue/dequeue requires a significant overhead that could make Create operations slightly slower and they are not always required, but that doesn't change the fact that such operations can be valuable to many systems, and there is a lot of space for improvements in that area.

At last, there is an open discussion about what are graph databases and what are their differences and similarities with the other databases. Graph databases have proven to apply in more than one from the known categories (SQL/NoSQL, etc), and they have unique characteristics that can define graph databases as a unique category. A theoretical frame about what is a graph database and what is not may be desired.

In conclusion possible topic for future research can be:

- Comparison of performance between Neo4j and MySQL
- Possible uses and improvements for BASE databases
- Performance analysis of GraphDB
- Use of Data Structures' characteristics on Databases
- Efficient FIFO and LIFO operations on Databases

- Definition of graph databases
- Investigation of the theoretical complexity of Graph Databases

References

1. R. A. A Comparison of Current Graph Database Models. In *IEEE 28th International Conference on Data Engineering Workshops*; 2012; Arlington, VA, USA. p. 171-177.

2. Ricardo CM, Urban SD. *Databases illuminated*. 3rd ed.: Jones and Bartlett Publishers, Inc; 2015.

3. Weiss MA. *Data structures and algorithm analysis in Java*. 3rd ed. Harlow, Essex: Pearson; 2012.

4. Kotsev V, Minadakis N, Papakonstantinou V, Erling O, Fundulaki I, Kiryakov A. Benchmarking RDF Query Engines: The LDBC Semantic Publishing Benchmark. In *CEUR Workshop Proceedings*; 2016.

5. Lopez FMS, Cruz ESDl. Literature review about Neo4j graph database as a feasible alternative for replacing RDBMS. *Industrial Data*. 2015; 18: p. 135-139.

6. Zhang C, Lu J. Holistic evaluation in multi-model databases benchmarking. *Distrib Parallel Databases*. 2021.

7. Jarrah M, Al-khatieb B, Mahasneh N, Al-khateeb B, Jararweh Y. GDBApex: A graph-based system to enable efficient transformation of enterprise infrastructures. *Softw Pract Exper*. 2021;: p. 517– 531.

8. Neo4J. *Neo4j Company Page*. [Online].; 2021]. Available from: https://neo4j.com/company/.

9. GNU. *GNU General Public License*. [Online]. [cited 2021]. Available from: https://www.gnu.org/licenses/gpl-3.0.html.

10. Ontotext. *Ontotext Company Page*. [Online]. [cited 2021]. Available from: https://www.ontotext.com/company/.

11. Ontotext. *Official GraphDB Page*. [Online].]. Available from: https://www.ontotext.com/products/graphdb/.

12. W3C. *RDF Standard by W3C*. [Online]. [cited 2021]. Available from: https://www.w3.org/RDF/.

13. W3C. *World Wide Web Consortium Homepage*. [Online]. [cited 2021]. Available from: https://www.w3.org.

14. W3C. *SPARQL Query Language for RDF*. [Online].; 2008 [cited 2021]. Available from: https://www.w3.org/TR/rdf-sparql-query/.

15. SAP. *OrientDB Community Edition Homepage*. [Online]. [cited 2021]. Available from: https://www.orientdb.org.

16. The Apache Software Foundation. *APACHE LICENSE, VERSION 2.0*. [Online]. [cited 2021]. Available from: https://www.apache.org/licenses/LICENSE-2.0.

17. SAP. *SAP Home Page*. [Online].]. Available from: https://www.sap.com/index.html.

18. Oracle. *Oracle Documentation, Database Concepts*. [Online]. [cited 2021]. Available from: https://docs.oracle.com/cd/B19306_01/server.102/b14220/schema.htm.

19. Neo4J. *Neo4J: Understanding Neo4j's data on disk*. [Online]. [cited 2021]. Available from: https://neo4j.com/developer/kb/understanding-data-on-disk/.

20. SAP. *OrientDB Documentation, PLocal Storage*. [Online]. [cited 2021]. Available from: https://orientdb.com/docs/last/internals/Paginated-Local-Storage.html.

21. W3C. *RDF 1.1 Concepts and Abstract Syntax*. [Online].; 2014 [cited 2021]. Available from: https://www.w3.org/TR/rdf11-concepts/.

22. Ontotext. *Ontotext: Fundamentals*. [Online]. [cited 2021]. Available from: https://www.ontotext.com/knowledge-hub/fundamentals/.

23. Neo4J. *Neo4J Manuals: Installation on Windows*. [Online]. [cited 2021]. Available from: https://neo4j.com/docs/operations-manual/current/installation/windows/.

24. Ontotext. *Ontotext: Quick Start Guide*. [Online]. [cited 2021]. Available from: https://graphdb.ontotext.com/documentation/standard/quick-start-guide.html.

25. SAP. *Download Page for OrientDB*. [Online]. [cited 2021]. Available from: https://orientdb.org/download.

26. OrientDB Community. *OrientDB Github Repository: Delete edge error: Cannot update record the resource has restricted access #6425*. [Online]. [cited 2021]. Available from: https://github.com/orientechnologies/orientdb/issues/6425.

27. SAP. *OrientDB Documentation: Database Security*. [Online]. [cited 2021]. Available from: https://orientdb.org/docs/3.2.x/security/Database-Security.html.

Appendix I: GitHub Repositories and Contact

The repository that the profiler is stored together with the files of the results and the legacy code can be found here:

https://github.com/Exarchias/graph-databases-profiler

Also, here are ways that you can get in contact with us:

Alm Robert: RobertKristianAlm@gmail.com

GitHub: https://github.com/Exarchias

LinkedIn: https://www.linkedin.com/in/exarchias/

Imeri Lavdim: lavdi-imeri@hotmail.com

GitHub: https://github.com/lavdimimeri

LinkedIn: https://www.linkedin.com/in/lavdim-imeri-446384214/