Tampere University

Lauri Korhonen

EXPLORING GRAPH DATABASES AND POSSIBLE BENEFITS OF UTILIZATION IN CONTENT SERVICES PLATFORMS
Case: M-Files

Master of Science Thesis
Faculty of Information
Technology and Communication
Sciences
Marko Junkkari
Mika Rantanen
April 2022

ABSTRACT

Lauri Korhonen: Exploring graph databases and possible benefits of utilization in content services platforms
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
April 2022

Interest in graph database technology has been raised by successful implementations of proprietary graph databases such as used by Twitter and Facebook as well as by emergence of general purpose graph databases. M-Files is a Content Services Platform (CSP) which mostly utilizes relational databases and expressed interest in graph databases. Could M-Files benefit from utilizing graph databases?

The context of CSP is first established via Enterprise Information Management (EIM). EIM helps in understanding why enterprises have information management needs and how information management software can solve related problems. Then databases used by leading CSP providers are presented to give an overview of the database technology used in leading platforms. Relational databases are the most used and therefore fundamentals of relational databases and graph databases are explained. Graph databases are explained in more detail and are presented with suitable use cases to give a good basic understanding of the technology. A comparison of these databases is presented to emphasize the strengths of graph databases. The strengths are emphasized to show that graph databases do excel with relationship rich data as is claimed. However, there are use cases in which a graph database is not a good option and instead a relational database should be used. These are also presented and help in understanding whether graph databases should be utilized or not when considering between options. Then M-Files is introduced along with graph database suitable use cases. Possible benefits of utilizing graph databases in M-Files are also presented. A proof-of-concept application which populates a Neo4j graph database with M-Files data was built. The application logic is presented along with used data modeling. Then the Neo4j graph database is compared to a relational database configuration used by M-Files. 5 different queries were each executed 10 times in both databases and the execution times were compared. The queries produced same results in both databases. The application shows that M-Files has some built-in readiness to adopt graph databases.

Utilizing graph database technology presents opportunities to innovate for M-Files by enabling a more personalized experience via deeper understanding of the data associated with M-Files. Graph database technology also presents architectural benefits for M-Files by being a viable option for cloud native architecture.

Keywords: Graph, database, content, services, platform

TIIVISTELMÄ

Lauri Korhonen: Graafitietokantojen ja mahdollisten hyötyjen tutkiminen sisältöpalvelualustoissa
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Huhtikuu 2022

Graafitietokannat ovat keränneet suosiota lähivuosina. Yksinoikeudella tuotettujen graafitietokantojen onnistuneet toteutukset kuten Facebookin ja Twitterin graafitietokannat, ovat herättäneet kiinnostusta. Toinen pääsyy kasvaneelle kiinnostukselle on yleiskäyttöisten graafitietokantaratkaisujen parantunut saatavuus. M-Files on sisältöpalvelualusta (CSP eli Content Services Platform) joka hyödyntää relaatiotietokantoja. Voisiko M-Files hyötyä graafitietokantateknologiasta?

Aluksi selvennetään yritystiedon hallinnan kautta konteksti ja mitä on CSP. Yritystiedon hallinta selventää miksi yrityksillä on tiedonhallinnan tarpeita ja kuinka tiedonhallintaohjelmistot ratkaisevat tähän liittyviä ongelmia. Sitten esitellään mitä tietokantoja johtavat CSP palveluntarjoajat hyödyntävät. Tämä antaa tilannekuvaa siitä, millaista tietokantateknologiaa johtavat palveluntarjoajat hyödyntävät. Koska relaatiotietokannat ovat yleisimpiä, esitellään ne ja graafitietokannat. Graafitietokannat esitellään yksityiskohtaisemmin, mikä antaa hyvän ymmärryksen kyseisen teknologian perusteista. Graafitietokannoille suotuisat käyttötapaukset esitellään. Kyseisiä tietokantateknologioita vertaillaan korostaakseen graafitietokantojen tiettyjä vahvuuksia. Tämä vertailu osoittaa, että graafitietokannat ovat tehokkaita suhderikkaan datan kanssa, kuten väitetään. Myös käyttötapaukset, jolloin relaatiotietokantojen hyödyntäminen olisi parempi vaihtoehto, esitellään. Tämä auttaa ymmärtämään mitä tietokantaratkaisua kannattaa käyttää, kun harkitsee relaatiotietokantojen ja graafitietokantojen välillä. Vertailun jälkeen itse M-Files esitellään. Sitten esitetään graafitietokantojen mahdollistamia käyttötapauksia, jotka soveltuvat M-Filesin kontekstiin. M-Files toteuttaa jo yhtä näistä käyttötapauksista relaatiotietokantojen avulla. Tähän käyttötapaukseen liittyen esitellään tehty kokeellinen sovellus. Kyseinen sovellus tallentaa M-Filesin dataa Neo4j graafitietokantaan, ja käytetty tiedonmallinnus esitellään. Kyseistä graafitietokantaa vertaillaan M-Filesin käyttämään relaatiotietokantaratkaisuun. Viisi erilaista kyselyä luodaan ja jokainen ajetaan 10 kertaa molemmissa tietokannoissa. Kyselyt tuottavat samat tulokset molemmissa tietokannoissa ja niiden suoritusnopeutta vertaillaan. Kyseinen sovellus osoittaa M-Filesin osittaista valmiutta omaksua graafitietokantojen käyttöä

Graafitietokannat tarjoavat M-Filesille mahdollisuuden innovoida. Kyseinen teknologia mahdollistaa personalisoidumman käyttäjäkokemuksen luomisen. Tämä onnistuu syvemmällä ymmärryksellä datasta, johon M-Filesilla on pääsy. Natiivin pilviarkkitehtuurin kannalta graafitietokannat ovat myös parempi vaihtoehto kuin relaatiotietokannat.

Avainsanat: Graafitietokannat, sisältöpalvelualustat, yritystiedonhallinta, sisällönhallinta

PREFACE

Thank you, M-Files, for providing an interesting topic. Thanks to Antti Nivala, Markku Laitkorpi, Juha Lepola, Mika Turunen, Minna Hautamäki, and others who participated in discussions on the topic as well as for guidance, feedback, trainings, and showing interest. Interest shown in the matter has had a surprisingly motivating effect. Special thanks to Ari Metsäpelto for helping me with the experimental application by shedding light on the option of using the *IObjectIndexer* interface and helping with the setup of the vault application. These people helped me to stay motivated and on the track. Also, thanks to my wife Siiri for the homefront support.

Finally, thanks to Marko Junkkari for the guidance and feedback through the thesis process, and to Mika Rantanen for the patience of reading through my thesis multiple times and for providing valuable feedback.

Kajaani, 27th of April 2022

Lauri Korhonen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ACL	Access Control List
ANSI	American National Standards Institute
BI	Business Intelligence
CODASYL	Conference on Data Systems Languages
CRM	Customer Relationship Management
CRUD	Create, Read, Update, and Delete
CSP	Content Services Platform
DAG	Directed Acyclic Graph
DB	Database
DBMS	Database Management System
DBTG	Data Base Task Group
EAS	Enterprise Application Software
ECM	Enterprise Content Management
EFSS	Enterprise File Synchronization and Sharing
EIM	Enterprise Information Management
EIS	Enterprise Information System
ERP	Enterprise Resource Planning
ES	Enterprise Software
ES	Enterprise System
IDS	Integrated Data Storage
IIM	Intelligent Information Management
IMS	Information Management System
ISAM	Indexed Sequential Access Method
JSON	JavaScript Object Notation
MDM	Master Data Management
NoSQL	Not only SQL
RDBMS	Relational Database Management System
REST	Representational State Transfer
SQL	Structured Query Language
VSAM	Virtual Storage Access Method
XML	Extensible Markup Language

1. INTRODUCTION

A digitalized enterprise may rely on multiple Enterprise Application Software. Combination of used software forms an Enterprise Information System which reflects an enterprise's information management needs. (Xu 2011; Niu *et al.* 2013) With multiple utilized software, for different functions of an enterprise, much data and information assets are created in the process. Information assets may be scattered in various locations such as in multiple databases and/or accessed by various software. Information may be slow to access or even inaccessible due to being saved in locations forgotten or unknown. As a solution to problematic information management, such as the challenge of integrating multiple sources of data from heterogenous software, Enterprise Content Management software were created to provide a centralized source of information. (Chaki 2015)

Recently (last decade), due to the evolution of enterprises' information management needs and software, it was deemed that Enterprise Content Management was no longer reflecting the needs. After an era of Enterprise Content Management, it was realized that successful information management requires also integrating the utilized Enterprise Application Software's functionality. Integrations provide more streamlined work experience. Enterprise Content Management software are morphing into Content Services Platforms. Shegda *et al*. (2016) define Content Services concisely and descriptively: "Content Services are a set of services and microservices, embodied either as an integrated product suite or as separate applications that share common APIs and repositories, to exploit diverse content types and to serve multiple constituencies and numerous use cases across an organization." This thesis follows Shega *et al.*'s definition.

M-Files Corporation develops a Content Services Platform - M-Files. M-Files mainly utilizes relational databases. Relational databases are the most used databases (DB-Engines 2021). Some database engine types offer advantages over relational databases in certain scenarios. In the context of Content Services Platform, a look is taken into a Not only SQL (NoSQL) database type - graph database, and the advantages it could offer for a Content Services Platform such as M-Files over relational databases. This is done mostly by a literature review. Graph databases have been around for multiple decades but have only recently gained attention.

The context of information management in enterprises is first established with the help of literature. This helps in understanding enterprises need to utilize a Content Services

Platform. Then relational databases and graph databases are explored. The fundamentals of the two database types are introduced. Graph databases are explained in more detail and suitable use cases are presented, as well as signs of use cases better suitable for relational databases. Relational database fundamentals are briefly introduced as a reminder, which should help in comprehending the fundamental differences of these two technologies. The differences are further emphasized by comparing data modeling, querying and performance by referring to online material and a publication. The performance comparison does not yield absolute truth about one engine being more performant than the other, but rather emphasizes the factors that impact performance. Graph databases can be utilized in a way, that they are not slowed down by the amount of data stored in it. This characteristic of graph databases may be found compelling, as the amount of data stored in a database can be associated with deteriorating performance which is often the case in relational databases.

The overview and comparison of relational databases and graph databases should convey perception of these technologies' maturation. Relational databases are well established, have a standardized query language and can be found used in all types of systems. Graph databases are more recent technology, though based on graph theory, which has a longer history than the relational model. Graph database technology is a powerful tool for certain use cases.

In this thesis the objective was to explore whether graph databases could present benefits for M-Files. To answer this question, the context of Enterprise Information Management was explored and a definition for Content Services Platforms was sought. This is done to gain understanding of the context and the purpose of M-Files. Understanding context is important in realizing what is relevant. The context of Enterprise Information Management is explained in the next Chapter.

2. ENTERPRISE INFORMATION MANAGEMENT

Xu (2015) defines Enterprise Information Management as follows: “Enterprise Information Management (EIM) is an integrative disciple for structuring, describing and governing information assets across organizational and technological boundaries to improve efficiency, promote transparency and enable business insight." The exact same definition is in found online in Gartner Glossary of EIM. EIM includes managing structured and unstructured enterprise owned information. Information is typically obtained from multiple sources. EIM software provides enterprises a way to manage information assets. Software that provides information management are called Enterprise Software (ES) or Enterprise Application Software (EAS). There are numerous kinds of Enterprise Application Software that serve their own function in an enterprise. Perhaps the most familiar examples of EAS to the reader are Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) or Business Intelligence (BI). EIM software focuses on information assets and their relevance grows in the ever-growing world of data, especially when multiple EAS are utilized. Via Enterprise Information Management, the need for a Content Services Platform is explained.

2.1 Enterprise information systems

Enterprise Information System (EIS) also known as Enterprise System (ES) is an integral toolset for modern digitalized enterprises. EIS provides integration and extension of business processes on both intraorganizational and interorganizational levels which contribute to streamlining business operations. EIS provides productivity increasing solutions to the growing needs of information integration in varying industries. EIS's emergence during the last two decades has been fueled by development in information technology and the need to integrate information in business operations. (Xu 2011; Niu *et al.* 2013) EIS comprises of the Enterprise Application Software that an enterprise utilizes. In other words, Enterprise Information System is a combination of the Enterprise Application Software an enterprise utilizes.

EAS such as ERP have mainly been developed to be utilized with physical assets of an enterprise since the 90s. Such software provides users quick access to information on enterprise's assets. Assets such as storage information, utilization of machines and human workers, material flow and operational efficiency. These examples are still relevant

IT requirements in modern industrial automation. (Niu *et al.* 2013) However, enterprises have begun also realizing the importance of information assets.

Few common EAS as earlier mentioned are ERP, CRM, and BI. ERP market is growing fast and is one of the most profitable areas in the software industry. (Xu 2015) ERP is an invaluable tool to handling company assets. CRM is used for managing relationships between a company and its current and potential customers. (Gebert *et al.* 2003) Relationship handling is critical in serving and gaining new clients. BI's objective is to enable businesses to make better decisions faster, convert data into information which is applied to create knowledge used for rational approach to management. (Vitt *et al.* 2008) BI tools such as Microsoft's Power BI exist to do such and to help generate visual reports and representations of data to help argue over business decisions (Microsoft).

Each Enterprise Application Software an enterprise utilizes as a part of their Enterprise Information System, such as the mentioned, generate information. Each EAS might have generated function specific information. Information is typically stored and reachable via the EAS that generated the information or stored in a place of choice by the end user. Customer relationship information is stored in the CRM application, generated reports are stored in the BI application storage and assembly line related information is accessible by the ERP application, or the users may have stored some information in a local storage or in a cloud storage. Accessing the information can be cumbersome as each EAS might have their own storage, or the location of the data is unknown or forgotten. The user must know what type of information is sought, the location, and must also be able to navigate through multiple heterogeneous systems. Enterprise Information Management software seek to solve related issues.

2.2 Enterprise content management

A central objective in Enterprise Information Management (EIM) is to define a strategy on the management of enterprise's information assets, decision making and execution. Information assets are not only limited to the business processes within an enterprise but can also relate to collaboration and supply chains. Thus, information assets can be both internal and external. There is a need to process different types of information. Varying standards define how certain information must be handled. Information lifecycle, sharing, storing etc. may be strictly governed. These are some challenges in information management that Enterprise Content Management (ECM) software sought to solve. (Chaki 2015)

EIM's roots are in the early 1990s with the rise of structured data. This was enabled by the development of relational database management systems. With systems such as Oracle, SQL Server and DB2 it had become easy to model, store and transfer structured data. Later the concept of managing structured data changed to include unstructured data as well. It was realized that about 80 % of all enterprise data is unstructured. This resulted in disciplines evolving such as ECM. (Chaki 2015)

ECM's objective was to help enterprises solve information management issues as information volume and complexity grow (Tyrväinen *et al.* 2006). ECM software provides a tool which solves issues related to capturing, processing, accessing, measuring, integrating, and storing of information. It offers control of all information through its lifecycle. Transitioning to using ECM could help enterprises enhance productivity, streamline processes, track assets, comply with regulations, eliminate redundancy in information storing and ease accessing information. (Hullavarad *et al.* 2015)

ECM can be beneficial businesses of varying sizes. Software which solves information management issues is beneficial in many ways. Standardized processes for handling information or so-called workflows assure that certain type of information complies to requirements which might comply to legal demands. For instance, workflows can help an enterprise assure that their business processes comply with quality requirements in an audit. Search tools can be enabled with scanning the contents of documents. This makes it possible for workers to search for data and information which can increase productivity. A central source of information eliminates the need to navigate through multiple heterogenous systems. These are only few examples of how ECM might have helped an enterprise execute information management strategies and enhance productivity.

2.3 Content Services Platform, ECM refined

ECM emerged as a discipline from Enterprise Information Management and evolved from handling structured data to also include handling unstructured data. However, ECM failed to deliver as such. As of today, ECM is no longer reflecting market dynamics and does not meet organizational needs for content in digital business. ECM has had to evolve beyond its original scope to meet enterprise business requirements. Gartner forecasted that 20 % of major ECM and Enterprise File Synchronization and Sharing (EFSS) businesses will morph their existing offerings into Content Services Platforms by 2020. (Shegda *et al.* 2016)

As an afterthought, it may not come as a surprise that ECM applications are now evolving into CSPs. It was concluded early on by Tyrväinen *et al.* (2006) that despite the interest in ECM, as it was adopted by software vendors and practitioners, the ECM discipline had received only little attention in the information systems research community. As ECM received wider adoption by enterprises, its need to evolve was recognized. ECM's need to evolve might have originated from the lack of research; failure to understand EIM needs when defining ECM. The evolution of Enterprise Information Systems has probably also affected the need to evolve; EIM needs have also changed. There are external forces and internal drivers that have reshaped the ECM market. New Enterprise Application Software emerge and thus EIS evolve and so does the requirements for EIM; data and volumes grow and so changes the requirements for solutions seeking to successfully implement Enterprise Information Management. (Shegda *et al.* 2016)

According to a recent survey by Gartner (2015) in which 2000 respondents in organizations with 100 or more employees, 27 % responded to use at least one application that they have obtained for themselves. The application was not approved by their organization. The respondents did this in search of good content experience outside of their ECM product suite. 20 % respondents used such applications to engage with customers. ECM as such had failed to respond to the business needs of managing content; business critical content which can be both formal and ad hoc as well as internal and external. (Shegda *et al.* 2016) A centralized ECM solution does not answer to business needs as cloud, social, mobile, analytics and digital business have changed ECM. A paradigm shift from ECM to Content Services has begun. (Hanns 2016)

Digital transformation is driving the paradigm shift. Enterprise Information Management has to do with more than centralized storing and accessing of business-critical information. New Enterprise Application Software are being introduced into the digital workplace. Content services are replacing ECM solutions as there are requirements for integrating with new EAS Application Programming Interfaces (APIs), platforms and components. (Hanns 2016) As Shegda *et al.* (2016) define Content Services: "Content Services are a set of services and microservices, embodied either as an integrated product suite or as separate applications that share common APIs and repositories, to exploit diverse content types and to serve multiple constituencies and numerous use cases across an organization." EIM needs of today's enterprises have redefined ECM to CSP.

2.4 Enterprise Information System data

The utilized Enterprise Application Software form the Enterprise Information System. Each EAS process's function specific data which can be obtained from various sources.

Fundamental characteristic of Content Services Platform is being able to integrate multiple EAS and streamlining the EIS. EAS have business critical roles in digitalized enterprises. Business critical data is obtained from various sources such as sensors, user input, interactions, intermediate storages, databases, scanners, other EAS etc. Databases are often used as the intermediate storage of data. After an EAS has received data, it is further processed into business-critical information which is also stored in a database used by an application. The process of EAS receiving data allows communicating real world phenomena to a system. It is typical of software to utilize database technology to store and retrieve data and information.

The heterogenous nature of EIS imposes varying needs for databases. EAS which process structured information has different requirements for a database than an application which stores unstructured data. For instance, an EAS which processes measurements such as sensor provided data, can utilize relational databases. A sensor measures a physical phenomenon and outputs data of strict structure. The structure is rarely changed due to the design of embedded systems. However, as applications and requirements change, there will be need to modify how information refined from data is stored. It might be hard to interpret structure in this information. Contracts, emails, reports, notes, and other information generated by human interaction are unstructured information, though follow a structure to a certain degree.

Concluding from most enterprise data being unstructured, storing data is not as simple as designing a database based on the structure of data. The problem concerning data has to do with of other concerns than simply storing the data of interest. In addition, information about processing the data must be stored; this is data governance, an important aspect of EAS. It is important to store information about the data's lifecycle. When storing information, the aspect of retrieving it should also be considered. A key feature of a database is the ability to efficiently retrieve and find needed data. When data and information become complex and large in volume, special consideration must be applied to how the storage is organized. Designing a database for EAS should be done in compliance to the application logic and business needs. Designing a database for CSP must support the integration of multiple heterogenous EAS and databases. EIS data is typically scattered in multiple and possibly redundant databases. For multiple decades, relational databases have been the main solution for EIS data needs. While relational databases serve the purpose of efficiently storing and retrieving data, graph databases have recently gained increasing amounts of attention and are changing the ways of how data is handled. In the next Chapter, the leading CSP offering is explored from database point of view.

3. DATABASES UTILIZED BY LEADING CSP PROVIDERS

Databases are essential in software that have a need to store data. In an application specific context, designing and implementing a database is done within the requirements imposed by the applications needs. CSP should enable Enterprise Application Software integrations and components that support business processes. Not only should these be supported by the database but also the ECM requirements are still relevant. Therefore, it can be argued that the flexibility of the database is an important requirement. The CSP provider cannot know all the application or component requirements beforehand, as these tend to change according to the dynamic nature of Enterprise Information Systems. Some CSP providers publicly share takes on architectural information which are presented. This allows a glimpse of the current standing of databases in CSPs. Found architectural information about the CSP providers which were classified as the leaders by Gartner are presented next.

3.1 Hyland

Hyland is rated in the Magic Quadrant as a strong leader, lacking behind only Microsoft in ability to execute (Woodbridge *et al.* 2021). Hyland owns multiple CSPs, which make up the Content Services offering. Nuxeo, Alfresco content services, OnBase and Perceptive Content are categorized as Hyland's Content Services Platform (Hyland 2021; Gartner). These CSPs use and support multiple different databases.

Nuxeo is an open-source Enterprise Content Management platform classified also as a CSP. Nuxeo has released a whitepaper (2021) Designing a Modern Platform Architecture for Content Services. NoSQL is emphasized as an important part of a modern CSP architecture. Their updated documentation states that "Nuxeo applications store most of their data in a SQL database. Several databases are supported, but they must be configured to work correctly." (Nuxeo) The same documentation shows that PostgreSQL, Oracle, Microsoft SQL Server, MySQL, MariaDB and MongoDB are supported. While in the whitepaper NoSQL is emphasized to be an important part of modern CSP architecture, its role remains unclear. Relational databases are used to store most of data, but NoSQL databases are utilized as well.

Alfresco's technical documentation part "Configure databases" (Alfresco) states that supported databases are Amazon Aurora, MySQL, Oracle, PostgreSQL, and Microsoft SQL

Server. The mentioned are also supported in Amazon Relational Database Service in the cloud. Additionally, also MariaDB is supported. Deducing from the supported databases by Alfresco, it seems that relational databases are mostly utilized and supported. While Oracle does have a graph computing solution, it is unclear if this is supported by Alfresco (Oracle).

OnBase is Hyland's flagship enterprise information platform (Hyland). According to KeyMark, which has been Hyland's OnBase's reseller since 1999, OnBase relies on SQL Databases. OnBase uses a proprietary database schema in which relationships between each table is controlled by software, rather than by an SQL server. The software offers indexing, and it is warned not to let the index get outdated or performance will deteriorate due to resorting to scanning of full tables. (Keymark) Hyland's Perceptive Content, a content and process management product suite, also relies on relational databases (Hyland; Manualzz).

It seems that Hyland, a leader in the Magic Quadrant, mostly utilizes relational databases in their CSP offering. This assumption is based on the available technical documentation. Nuxeo is the only mentioned CSP in this context which addressed utilizing NoSQL databases. OnBase also supplements their relational database with proprietary software for better performance.

3.2 Microsoft

Microsoft is the leader in Gartner's Magic Quadrant for Content Services Platforms (Woodbridge *et al.* 2021). There is no need to search technical documentation and base assumptions on what database type Microsoft seeks to utilize in their Content Services. Microsoft's current CEO Satya Nadella called Microsoft Graph (formerly known as Office 365 API) the company's "most important bet" (Bisson 2021). Microsoft's CSP whitepaper gives the impression, that Microsoft Graph is used for multiple purposes, such as for a knowledge graph which provides Sharepoint users recommendations, for automatic metadata and for Artificial Intelligence (AI) which applies knowledge mining. Sharepoint is the backbone content services layer in Office 365. Graph connectors enable Microsoft Search Service, which can be used to discover content for search capabilities from various data sources. (Microsoft)

Microsoft utilizes their proprietary graph technology for multiple purposes. Microsoft Graph also exposes REST API's and libraries which can be used to access data on multiple Microsoft cloud services. (Campos *et al.*) Microsoft is very interested in graph-based data. Microsoft purchased LinkedIn and have begun showing LinkedIn data in tools such as Outlook. LinkedIn is based on a huge graph database. In addition, they have multiple other graphs; Microsoft Dataverse, Cosmos DB and Security Graph. (Microsoft) Microsoft Dataverse is formerly known as Common Data Service (Peart & Coulter 2021).

3.3 Box, OpenText

Box is a CSP which is also rated as a leader in the CSP Magic Quadrant, lacking behind Microsoft and Hyland (Woodbridge *et al.* 2021). In 2020, Box was named Gartner Peer Insights Customers' Choice vendor for CSP and Content Collaboration Tools (Box 2021). Box uses Apache HBase, also known as the Hadoop database, which is a distributed, scalable big data store, for storing documents. Documents are sent for indexing to Apache Solr clusters. Solr is an enterprise search server which is built on Apache Lucene. It is apparent that Box's approach to indexing and storing is focused on heavy scalability and centrality. In 2017, Box improved their indexing and were able to index almost 50 billion documents in less than two days in 2017. The system was capable of consistently conducting over 300 000 document reads per second. (Iqbal 2021; The Apache Software Foundation 2022) This level of performance is made possible by big data solutions. However, it must be noted the history of Box as an Enterprise File Synchronization and Sharing (EFSS) provider, which differs from ECM. (Basso *et al.* 2016) Concluding by their architectural decisions, it seems that their focus has been on scaling and supporting centralization. This probably has to do with creating a good experience on collaboration with external stakeholders of an enterprise, a focus area in Content Services Platforms, something that traditional ECMs failed to deliver on.

OpenText is a leader in the magic quadrant and has a proprietary relational database OpenText Gupta SQLBase (OpenText). In addition, they offer upgradeability and connectivity to multiple relational databases such as Oracle, Microsoft SQL Server, Microsoft Azure SQL DB, and IBM Db2. (OpenText) From the supported database connectivity and proprietary database technology, it seems OpenText utilizes relational databases in their information management.

3.4 Summary of databases used in leading CSPs

The leading CSP providers database technology solutions were briefly investigated. This was done by looking up technical documentation and other available information. Companies rarely keep their accurate architecture publicly available. This implies that this information is not accurate and does not depict the whole truth. However, it seems like majority of these inspected providers utilize relational database technology.

Relational databases are standardized and have been in wide use for several decades. The history behind their status is opened in the next Chapter. As these databases are transactional, optimized for running concurrently quick read- and write -operations, they are a good choice for CSPs. Though NoSQL addresses some of the issues of relational databases, the leading CSP providers mostly utilize relational databases with a few notable exceptions.

Nuxeo addresses the advantages that NoSQL offer and their whitepaper (Nuxeo 2021) states that NoSQL enables great performance benefits and scalability. Still, their documentation gives the impression that relational databases are still mostly utilized. Box utilizes big data solutions, which supports their architecture of centralization. Big data solutions are an overkill for CSP vendors who don’t seek to implement a large, centralized data storage. Microsoft as the recognized leader by Gartner is focused on developing their proprietary graph technology solution.

4. RELATIONAL DATABASES

Relational databases are well established in the IT industry. Relational databases follow the Relational Model which was developed by Dr. E. F. Codd in his research paper - *A Relational Model of Data for Large Shared Data Banks*. The paper released in 1970 is considered groundbreaking considering how widespread relational databases are. Structured Query Language (SQL) is the de facto standard for interacting with relational databases. It was standardized by the American National Standards Institute (ANSI) in 1986 and has undergone multiple revisions. (Batra 2018) ISO/IEC 9075 standard: "Information technology - Database languages - SQL" describes SQL. Latest revision is ISO/IEC 9075-15:2019 which adds multi-dimensional arrays. (ISO/IEC 9075-15:2019) This Chapter explores the history of the relational database, the most popular type of database to this day and its fundamentals. (DB-Engines 2021).

4.1 History of relational databases and SQL

In the late 1940s and 1950s computers were considered as advanced calculating machines. Soon, businesses realized their importance in automated processing and keeping of records. Early computers used tape storage for storing data. Accessing data was therefore restricted to sequential scanning of the records. With advancements in storage technology and the invention of magnetic disks and tapes, more sophisticated access methods were developed. IBM developed the Indexed Sequential Access Method (ISAM) which became prominent in the 1960s and its successor Virtual Storage Access Method (VSAM) as file-oriented technologies allowed key-based direct access for their mainframe systems. By the 1960s, file-oriented systems were used increasingly by computers. Then popular programming languages like COBOL and PL/I were used for these systems. (Batra 2018) As a sidenote about COBOL's widespread; In 2017 Reuters reported that 43 % of banking systems are built on COBOL, 220 billion lines of COBOL code are still in use and 95 % of ATM swipes rely on COBOL code (TIOBE Index *et al.* 2019).

New access techniques increased throughput, but they did not address the reality of data being split in multiple, independent files with no centralized logical structure. Redundant columns increased storage costs and data was typically inconsistent. This problem was addressed by logical data models and database systems to some degree. Information Management System (IMS) was released by IBM in 1966 for use in the NASA Apollo program. IMS was a hierarchical database which assumed that all data relationships could be structured as hierarchies. IMS was a leap in data modeling which implied proper thought on structuring data. Another data model which gained popularity in the 1960s was the network data model. It allowed representing data as a complex network with entities referring to each other. This was a more natural way of modeling data. Data which was being generated in business processes. One successful implementation of the model was the Integrated Data Store (IDS). It had such an impact that its design largely influenced the network model standard CODASYL DBTG. (Batra 2018) "The acronym DBTG refers to the Data Base Task Group of the Conference on Data Systems Languages (CODASYL), the group responsible for standardization of the programming language COBOL." (Thakur)

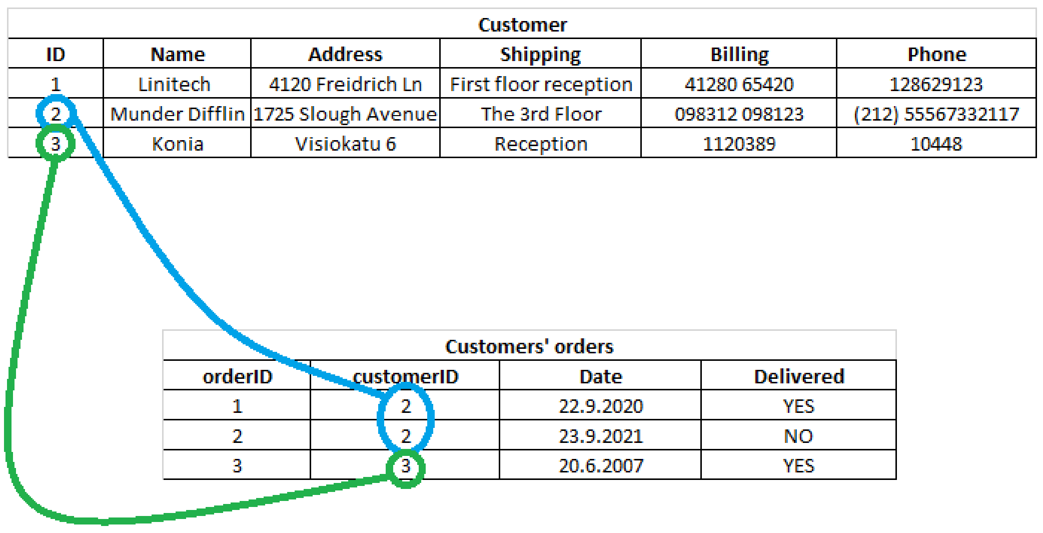
Hierarchical and network model was driven from the viewpoint of a programmer with knowledge on interconnecting the entities being modeled. Queries could only be run as envisioned by the programmers. However, Dr. Codd's proposed relational theory based on the belief of natural, logical relationships manifesting themselves when the domain of the data is understood. System should be ready for flexible querying. This model was built on mathematical theory rather than on programming. IBM Research Labs at San Jose, where Codd worked in the '70s and '80s gave birth to relational databases. IBM came up with the SystemR project which was a prototype relational database management system. However, IBM failed to realize the value and it was Relation Software Inc. that created the first widespread Relational Database Management System (RDBMS) in 1979 by borrowing IBM's research on SystemR. Relational Software Inc. is what is known today as the Oracle Corporation. (Batra 2018) Oracle's relational database engine is the most popular database engine as of today (DB-Engines 2021).

As mentioned, SQL is the de facto language for interacting with relational databases. However, SQL wasn't the dominant query language early on. Codd had proposed two languages for querying and manipulation of data - *relational algebra* and *relational calculus*. They were mathematical notations rather than query languages. Codd attempted to make a real query language, Alpha. Alpha was proposed in Codd's 1971 paper "A Data Base Sublanguage Founded on the Relational Calculus". SystemR however used a separate query language *SEQUEL* created by Raymond Boyce and Don Chamberlin around 1973. *SEQUEL* was eventually renamed to SQL. The Ingres project created *Alpha* influenced query language called *QUEL* but as in the '80s dominant vendors were pushing SQL, *QUEL* failed to gain a place on the industry. By the late '80s, SQL had gained a firm position as the de facto database query language which holds to this day. (Batra 2018)

4.2 Structure

Relational databases store and provide access to data points that are related to one another. Data is represented in structured tables which consist of rows and attributes. Each row is identified by a key consisting of one or several attributes. Usually, the key is a unique ID column. These unique identifiers are known as keys. Columns hold attributes of the data. Typically, each record has values for each attribute, but this is not required. Tables can have a varying structure and can represent different data. For instance, a customer information table contains records, each which represent a single customer and their information. A record could then include a name, address, shipping and billing information, phone number and other information as well as the key. A second table can store customers' orders. In this second table, a record contains the unique ID of the customer and order related information. As these two tables' records have the key in common, there exists a relationship between these tables. If the structure of the database needs to be changed, a database upgrade must be done. In the upgrade, the content of the database must be changed so that the records comply to the changed structure. During the upgrade, the database is unavailable.

To query the orders of a certain customer, the unique ID of the customer is used to query the order records with the same unique ID from the table containing customers' orders. (Oracle) Note that the orders also each have their own unique ID. The unique ID which is used to identify a record in a table, is called a primary key. The unique ID which is used to identify a relationship to another table, is a foreign key. Figure 1 depicts the relationships.



Customer

ID	Name	Address	Shipping	Billing	Phone
1	Linitech	4120 Freidrich Ln	First floor reception	41280 65420	128629123
2	Munder Difflin	1725 Slough Avenue	The 3rd Floor	098312 098123	(212) 55567332117
3	Konia	Visiokatu 6	Reception	1120389	10448

Customers' orders

orderID	customerID	Date	Delivered
1	2	22.9.2020	YES
2	2	23.9.2021	NO
3	3	20.6.2007	YES

Figure 1. *Depiction of related data between two tables.*

The order information records are "labeled" using foreign keys of the customers stating the relationship of these orders. The customer has a primary key ID, and the order records refer to that key as the foreign key *customerID*. In figure 1, in the *Customers' orders* table, can be seen that the order records that belong to a certain customer are identified by the same key from the *Customer* table indicating the relationship. Operations in which related information are sought from multiple tables are called joins. There are different types of join operations. Looking up information from tables are sped up by indexing, which enable looking up records sorted by attributes of interest. A good index allows faster lookup of information whereas no indexing or bad indexing will cause a slow join or a search of records in which the whole table must be scanned thoroughly. Good indexing is critical in high performing relational databases.

It is a common misconception that *Relational* in Relational Databases would refer to the relationships in the databases. This is not the case. The relational model means the separation of logical data structures and physical storage structures. The physical data storage can be managed without affecting access to that data as a logical structure. The distinction also applies to database operations. Actions that enable manipulation of data and structure are clearly defined. Logical operations allow specifying of the desired content in application context and physical operations determine how the content should be accessed and carries out the operation. For the data to be correctly accessed, referenced, and modified, certain integrity rules apply. A database administrator can apply different integrity rules, for example prohibiting duplicate rows reduces the possibility of erroneous information existing in the database. (Oracle)

4.3 ACID properties

Relational databases are used in organizations of all types and sizes for varying information needs. Inventories, ecommerce transactions, huge amounts of mission critical customer information, registries, banking software etc. have been built on top of relational databases. Relational databases can be considered for any information needs in which data relates to each other and require secure, rule-based and a consistent way of management. Relational databases support data consistency, in which multiple instances of a database should have the same data all the time. An example of this could be a customer depositing money at an ATM and then checking balance immediately after. Business rules and policies are handled with strict policies regarding permanent changes to the database. If an inventory database tracks that certain three parts are used together, it will not allow pulling only a single part from the database. The two other parts must also be pulled along with the third part before the database makes any commitment. This multifaceted commitment capability is called atomicity, which is the key to ensuring database being accurate. *Atomicity* ensures compliancy with the regulations, rules, and policies that a business should comply to. (Oracle) Atomicity is one of the four fundamental properties of relational databases, known as ACID. ACID is short for Atomicity, Consistency, Isolation, and Durability.

Consistency constrains defined on the data should be preserved by a transaction. When the changes are committed, all defined integrity constrains must be satisfied. Breaking the defined integrity constrains are allowed during the operations, but both the state from which data is and the state to which the transaction ends should satisfy the integrity constrains. The application should be programmed in such a way which ensures the consistency preservation. (Vossen 2009)

Transactions should be *isolated* from other transactions. Each transaction should behave as if there were no other transactions. Each transaction will deal with consistent data. If two transactions were to operate on the same data unit, the latter transaction would only see the committed data of the first transaction and never the effects of an incomplete transaction. *Isolation* is a decisive property which ensures success in programming concurrent operations. Transactions should behave as if they were sequential when executed concurrently. (Vossen 2009)

Data servers which are updated via transactions, and which notify changes being successful after a commit, are guaranteed to withstand failures. A relational database is *durable* so that it can survive hardware or subsequent software failures. (Vossen 2009) This is typically achieved by upkeeping a log file of transactions and with redundant data or backups. Depending on the logic of upkeeping the log, rollbacks may have to be executed in-order for the database to be able to return to a consistent state. Only consistent states are snapshot in compliance to atomicity and consistency. However, it is possible to implement snapshotting uncommitted transactions. When the database is returned to a running state, transactions are continued. Redundant data storages ensure durability when the physical memory containing the data is damaged.

Building a relational database and an application accessing it in compliance to ACID properties ensures runtime integrity of data. ACID properties can also be too restrictive for businesses. For long running operations in case of multiple transaction on the same data, it might be too slow. In these cases, alternative or additional guarantees need to be employed. (Vossen 2009) There are varying policies in relational databases which dictate what data transactions see, when tables or records can be accessed, and which determine what kind of a system is built. Systems such as batch processing, real-time, data warehouse and timesharing systems exist, but specific details are omitted. (Bernstein & Newcomer 2009)

4.4 Summary of relational databases

Relational databases have a relatively long history in the context of information technology dating back several decades. During this time, the technology has had time to mature. Multiple generations of programmers, software architects, database administrators and other stakeholders have been solving related problems with relational databases. There is extensive amount of knowledge available varying from books to scientific articles, forum posts, conferences etc. Knowledge that can be obtained to help avoid known pitfalls, accelerate implementation and to help build quality products. Not only due to the amount of knowledge available, are relational databases a viable option for building application data and information management solutions, but because they still hold up well technologically. Great amount of software relies on relational databases.

Building an application and implementing a highly performing well-modeled relational database requires good understanding of the technology. There are several factors which have an impact on the performance of the system and on the development lifecycle. Some aspects of relational databases not considered in this brief overview are optimization of queries, procedures, data modeling and normalization, etc. There are many factors to consider. The objective was to highlight key features and characteristics of relational databases.

5. GRAPH DATABASES

Graph database is a type of NoSQL databases. Graph databases are based on graph data models. Graph data models originate from "Graph Theory" which in mathematics is the study of graphs. The theory can be applied to solving problems in varying disciples such as architecture, social relationships, machine learning, medicine, geography, etc. and in this case, databases in information systems. In this Chapter, focus is on a certain type of a graph database model, labeled property graphs. There are multiple types of graph models in which real-world entities and their relationships are represented using a collection of conceptual tools. (Angles & Gutierrez 2008)

5.1 History of graph databases

Graph Theory, in which graph databases are based on, originates back to the 1700s. Swiss mathematician Leonhard Euler wrote an article about the city of Konigsberg. The Pregolya River passed through the city and seven bridges were situated across the river as depicted in figure 2.

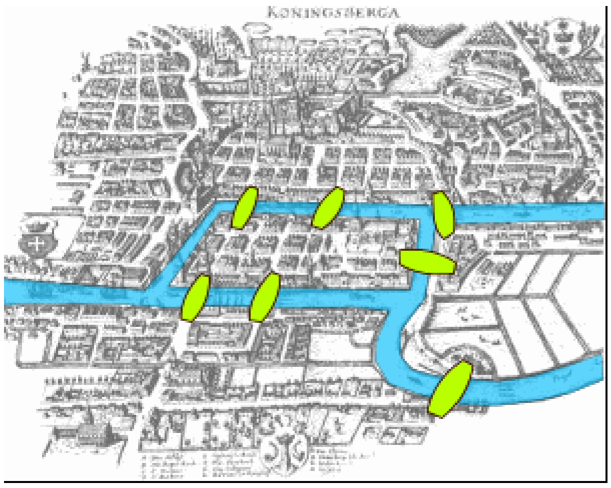


Figure 2. *The seven bridges in Konigsberg* (Bogdan Giuşcă 2005).

The citizens of Konigsberg realized that how hard they tried; they could not stroll a route in which they crossed each bridge exactly once. What Euler wrote on the Konigsberg Bridge Problem, is considered the beginning of the graph theory field. Graph theory itself was only found useful in solving puzzles and in analyzing games etc. and it was not until the mid-1800s people began to realize the usefulness of it in modeling things that were in the interest of society. (Kumar & Pattnaik 2018) To this day, new use cases in which graph theory can be applied to are being researched or implemented, such as the usage of graph database models in biology, fraud and anomaly detection with machine learning, knowledge mining, and analysis of wireless cyber-physical systems (Songqing *et al.* 2020; Prusti *et al.* 2021; Magomedov *et al.* 2018; Lei *et al.* 2020; Kashef 2021).

Graph database models began to gain attraction in the 80s and early 90s alongside object-oriented models. The first half of the 90s was most active on the topic. With the emergence of other database models, in particular Extensible Markup Language (XML), semi-structured, spatial, and geographical, the interest gradually died out. XML caught the attention of those working on hypertext. People shifted from working on graph databases to applications such as spatial data, Web, and documents. For most applications, the tree-like structure was enough. (Angles & Gutierrez 2008)

General purpose graph databases are a reality today. Consumers can experience the benefits of connected data and application programmers have the option to graph databases without the need to build their own graph infrastructure. (Robinson *et al.* 2015). Graph databases are also trending now. Calculated by number of mentions of the system on websites, general interest by Google Trends, frequency of technical discussions about the system on well-known IT-related sites Stack Overflow and DBA Stack Exchange, job offers, mentions in professional network profiles and relevance in social networks, graph databases are the most popular as seen in figure 3. (DB-Engines 2021)

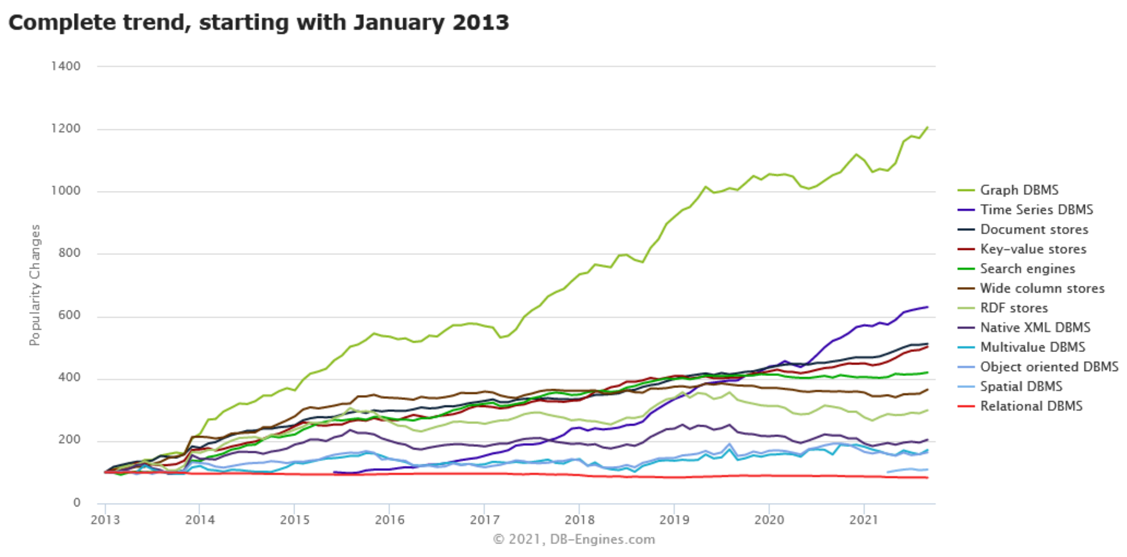


Figure 3. *Normalized DBMS popularity changes per category, starting with January 2013* (DB-Engines 2021).

By the time of this writing, Neo4j is by far the most popular graph Database Management System (DBMS) according to popularity ranking. Neo4j is almost 50 % more popular than the second most popular graph DBMS Microsoft Azure Cosmos DB which is a multi-model DBMS. (DB-Engines 2021) Neo4j is utilized by 75 % of the Fortune 100, all the North America's top 20 banks, eight of the top ten insurance companies and many other big companies (Neo4j Inc.). The technology seems to have been proven useful and is adopted by big and impactful enterprises. However, it is unclear how they utilize graph databases. This increased focus and success to adapt graph databases is mainly driven by the success of companies who have centered their business around their own proprietary graph technologies; Facebook, Google, and Twitter; and by the introduction of general purpose graph database technologies (Robinson *et al.* 2015). The graph technology landscape is blooming as can be seen in figure 4 on the next page. There are multiple graph databases and tools for varying purposes based on graph technology.

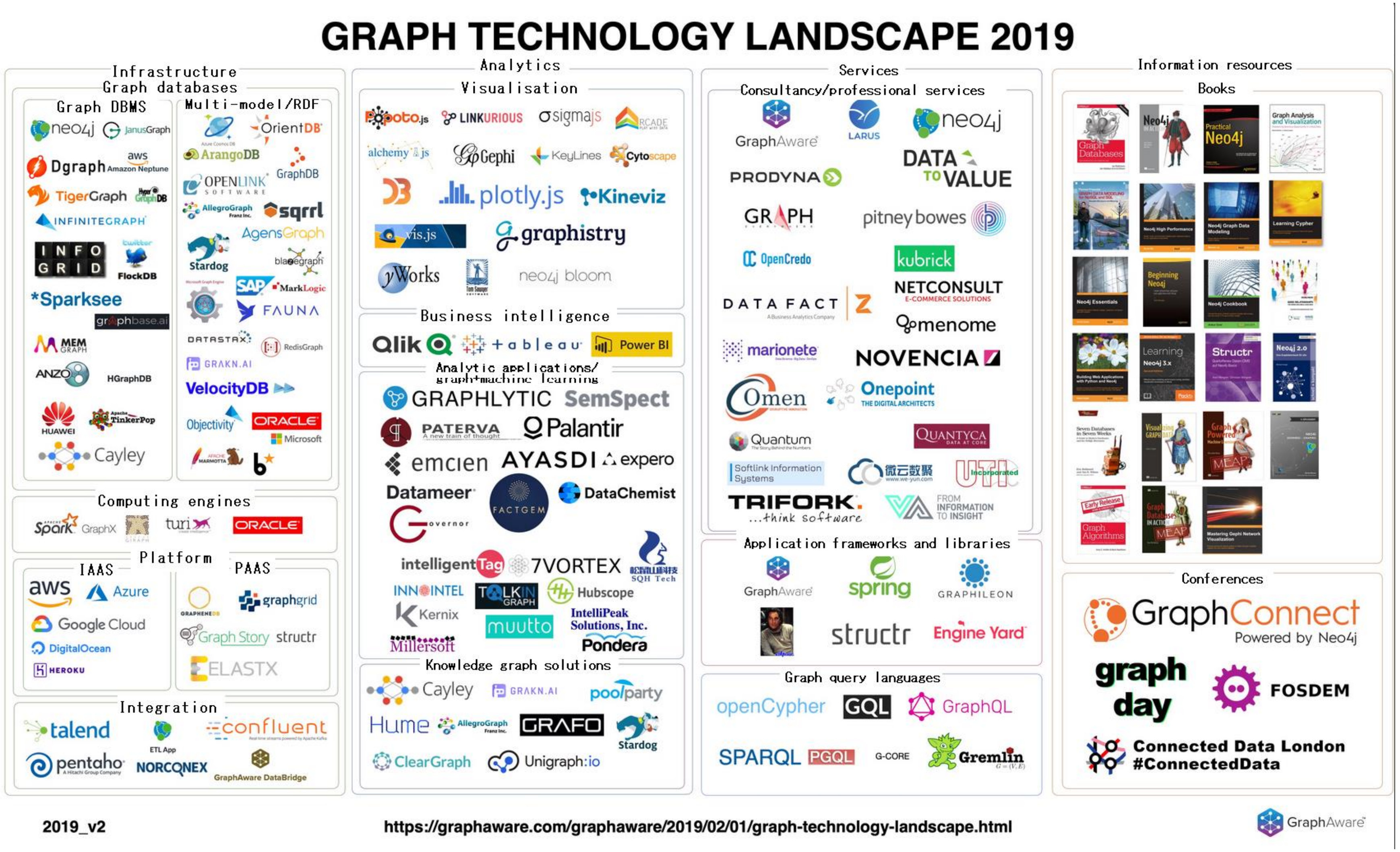

GRAPH TECHNOLOGY LANDSCAPE 2019
Infrastructure
Graph databases
Graph DBMS
Multi-model/RDF
Computing engines
Platform
IAAS
PAAS
Integration
Analytics
Visualisation
Business intelligence
Analytic applications/ graph+machine learning
Knowledge graph solutions
Services
Consultancy/professional services
Application frameworks and libraries
Graph query languages
Information resources
Books
Conferences
2019_v2
https://graphaware.com/graphaware/2019/02/01/graph-technology-landscape.html

Figure 4. *Graph technology landscape in 2019* (Szendi-Varga). (Original figure has been modified with larger category texts)

Worth noting is that there are multiple graph query languages available as there is no standard language. Choosing a specific graph DBMS also affects the query language choice. Neo4j and Microsoft Azure Cosmos Graph Database offer graph database technology categorized as labeled property graphs. What labeled property graphs are and what do they offer, is explored in the next Subchapter. As figure 4 contains some hard to read text, the full list of participants is made available in spreadsheet format by Szendi-Varga: https://github.com/graphaware/graph-technology-landscape/blob/master/Graph-TechnologyLandscape.csv.

5.2 Labeled property graphs

A graph is a collection of vertices and edges. In many references, vertices go by nodes and edges by relationships. These names are interchangeable. (Robinson *et al.* 2015) One advantage of graphs is being simple to illustrate as seen in figure 5. Diagrams of graphs usually consist of circles and lines representing nodes and edges as shown. (Perryman & Bechberger 2020)

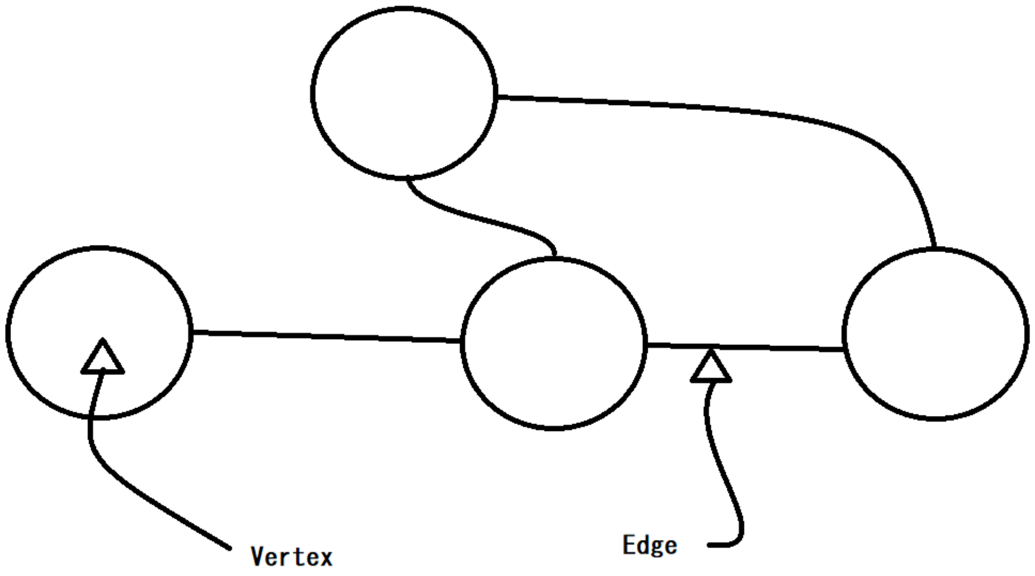


Figure 5. *A simple graph illustrated with circles as nodes and lines as edges.*

Graphs are useful in understanding a wide diversity of real-world datasets. Graphs are natural for representing many of the world's phenomena and once graphs are understood, they can be seen everywhere. (Robinson *et al.* 2015) Gartner identifies five types of graphs to be understood in the business world which should provide "sustainable competitive advantage": social, intent, consumption, interest, and mobile graphs (Valdes 2021).

Let's look at a social graph which can be used to represent Twitter's data. Figure 6 is an example of a labeled property graph. Each node is labeled as "User" which indicates role in the network. User nodes are connected by directed edges with a label indicating the relationship.

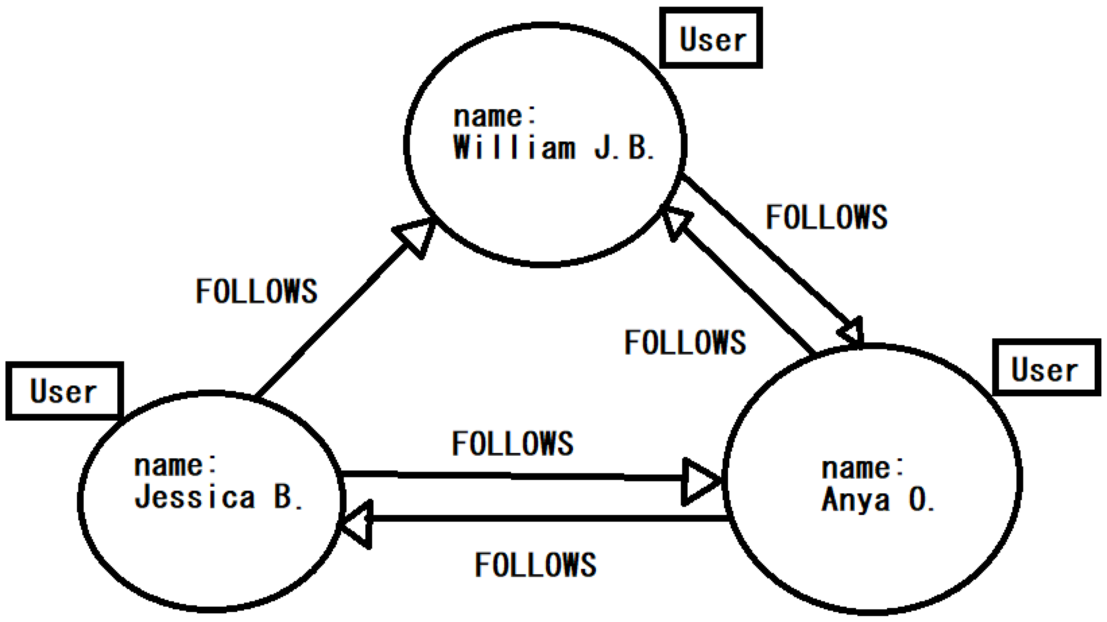


Figure 6. *A small social graph consisting of three nodes and five directed edges.*

In figure 6 are three nodes, each of them has a name property. Labeled directed edges indicate who follows whom. William follows Anya and vice versa, same with Jessica and Anya, however Jessica follows William, but William doesn’t follow Jessica. Twitter's real graph would be magnitudes larger, with many more properties, but the basic principles precisely apply. From this graph, it would be possible to query for example from William's point of view, who are the users who follow the same people as he does, whom he is not following. In this case the answer would be Jessica. In a large social graph, queries such as this could be used to create suggestions who to follow based on users with common interests.

Commonality can be used to create recommendation engines using graph databases. Graph would consist of users or user profiles who are interested in categorized products. Users and products would be nodes. Products then have category as a property or an edge to a category node. Users could have interest edges towards a product based on viewing items. Then users would be recommended products based on what kind of products and categories other users with similar interests have bought or viewed. Categories can be used as boundaries, as there might be shared interest among products in a category. However, users might not share other common category interests. Same principle could be applied to media streaming services or vacation booking services. Recommend movies to horror movie fans that other horror fans have liked. Liking a movie could be an edge constructed upon liking a movie. Recommend resorts based on what other users have viewed who have common resorts of interest. Apply boundaries of price, booking date, activities available etc. to make the recommendations more relevant.

Besides recommendation engines and social networks, there are multiple other use cases such as fraud or anomaly detection, knowledge graphs, network/asset monitoring, identity and access management, machine learning, protein-cell regulation, banking, etc. For each of these solutions, specific queries can be applied to the graph to obtain valuable information. Queries are often interchangeably called traversals in graph databases. To detect weird patterns or behavior and possible fraudulent activity in a banking graph, where company accounts are depicted as nodes and wire transfers are depict by edges, *follow the money*. Transitive relationships can be traversed to detect patterns such as large sums of money being circulated back to a single company via varying paths by companies who don't seem to have other things in common.

Labeled property graphs are a good general purpose graph data model. Various of problems can be solved using labeled property graphs and research is being conducted to find more use cases in different disciplines. The model is the most popular and has proven commercial success.

5.3 Graph database management system

Graph DBMS or graph database is an online database management system with Create, Read, Update, and Delete (CRUD) methods which are used to expose a graph data model. Generally, graph databases are built for use with transactional systems which allow executing concurrently several transactions. Graph databases are normally optimized for transactional performance with transactional integrity and availability in mind. When examining graph database technologies, two properties should be considered: The underlying storage and the processing engine. (Robinson *et al.* 2015)

Some graph databases use optimized native graph storage designed for storing and managing graphs, whereas some serialize the graph data into a relational database, or other general purpose data store (Robinson *et al.* 2015). Relationships are classified as first-class citizens in the graph data model, but serialization might slow exploring the relationships. Relationships are considered as important as the data points themselves. In an optimized graph database traversing edges is a constant time operation enabled by using pointers in memory.

Closely related to the underlying storage, the processing engine is the other important property to examine. Some definitions require graph databases to use index-free adjacency. (Robinson *et al.* 2015) Index-free adjacency means that each node is directly linked to its neighbor node. In a database engine utilizing index-free adjacency each node acts as an index of other nearby nodes. This is much cheaper than using global indexes. It makes traversals performant in which a starting point (or starting points) can be limited effectively, avoiding the need to perform massive traversals. Then relationships that satisfy conditions are traversed; physical pointers are directly dereferenced. In RDBMS there would be need to join tables for each relationship. (Pokorný 2015) In slightly broader terms, any database that behaves like a graph database from the user's perspective, qualifies as a graph database (Robinson *et al.* 2015).

It is important to acknowledge the underlying storage as a graph database which does not utilize index-free adjacency may not provide the desired processing power which is associated with graph databases. It must be emphasized that a graph database which does not utilize index-free adjacency has nothing to do with being good or bad as there are trade-offs in both cases. Using physical pointers and dereferencing offers good performance in limited traversals with the trade-off of making huge traversals costly.

A graph compute engine enables global graph computational algorithms to be run. This is typically run against a large dataset due to the nature of graph databases. These engines are designed to do many things such as identifying clusters in the data and answer questions such as how many common friends each friends have, who are the involved parties in each project, who is the most common actor whom with other actors have worked with etc. Due to the emphasis on global queries, graph compute engines are normally optimized for scanning and processing large amounts of information in batches. On the other hand, some graph compute engines concern themselves strictly on working with data from external sources and return the results for storage elsewhere. (Robinson *et al.* 2015) It is important to note that a graph compute engine is different from a graph database. A graph compute engine is optimized for running global queries and algorithms, while a graph database may be optimized for sub-graph traversals.

When considering a graph DBMS for good performance, it is important to consider how the graph DBMS is implemented. Serializing data to a relational database has some tradeoffs, as does the implementation of a processing engine. Index-free adjacency should provide fast traversals with the trade-off of making huge traversals costly.

5.4 Graph database aspects to consider

Even though just about anything can be modeled as a graph, it does not justify migration to graph databases in existing projects or choosing graph databases in a new project. Our pragmatic world of budgets, project schedules, standards and commoditized skill-sets require justifying from effort put into versus value gained point of view. Replacing a well-established and well-understood data platform requires other justification than the power and ease of data modeling. (Robinson *et al.* 2015)

One reason for choosing graph databases can be the acknowledged performance gain when handling connected data. An example of great performance on connected data is presented in Subchapter 6.2. Join-intensive queries deteriorates performance with the growth of datasets in relational databases. In graph databases the queries can be localized to a portion of a graph so with the growth of datasets, queries can remain relatively constant. Execution time in a graph is proportional to the size of the part of the graph traversed. (Robinson *et al.* 2015) This emphasizes the importance of acknowledging the performance impact that the modeling of the graph has. *What are modeled as entities and what kinds of relationships are created?* The modeling affects how the graph is traversed. This implies that the modeling should be done in compliance to application logic. Unnecessarily large traversals should be avoided being slow and costly. For running global queries and algorithms, a graph compute engine would be a better option.

Graph databases are schema-free, and an easily testable API is often provided which can contribute to faster development time. In contrast to relational databases, there is no need to deal with database upgrades. This implies that there should be governance over how the graph database is utilized. (Robinson *et al.* 2015) Governance should ensure that changes to a database structure are aligned with application logic. In addition to an easily testable API, a graph database can enable ease of writing automated tests and manual testing as constructing the needed sub-graph may be less restricted due to rejecting ACID properties. A graph database can be populated with partial data. This doesn't mean that graph databases don't have constrain features.

Not all commercial graph databases allow querying using a declarative language. This implies that these graph databases lack ability to optimize queries. (Pokorný 2015) The two most popular graph DBMSs Neo4j and Microsoft Azure Cosmos DB's (Database) graph databases support declarative querying. Cypher is Neo4j's query language which is a declarative language (Neo4j Inc.). Cosmos's graph database is queried using Gremlin, which can be written in either imperative, declarative or in a hybrid manner, containing both imperative and declarative aspects (The Apache Software Foundation). The language support should be considered when deciding the used graph DBMS.

Extracting data from non-graph data sources to a graph database with good modeling might not be a simple task (Pokorný 2015). The relationships in the original data source can be used to create edges between data points (nodes), if available. However, the structure of the relational database might not be optimal for use in the graph database. It is worth nothing that relationships are first-class citizens of a graph database, and the performance is based on the ease of traversing relationships in relationship rich data.

Different graph DBMS offer varying features and support. Comparison between viable DBMSs would be advisable before committing. The requirements for the database should be known before comparison. There is no standard yet for graph databases which might imply that DBMSs can offer significantly varying experiences.

Graph databases can offer performance benefits with highly connected data when provided with good data modeling. Graph databases are not associated with good for queries that span the whole database. Utilizing graph databases' potential requires requirements engineering, data modeling, gathering knowledge of the application domain, investigation of graph DBMSs features, and data governance plan. The flexibility of graph databases should aid when difficulties are encountered.

5.5 What should be solved with graph databases?

It is important to understand what kinds of problems should be solved with graph databases. This helps in recognizing problems that might be suitable for solving with graph databases. When inspecting graph database material, it is common to encounter broad statements of everything being a graph problem. Because something can be modelled as a graph, it doesn't mean that it should be. (Perryman & Bechberger 2020) Let's present some answers to what kinds of problems graph databases are suitable for solving and what not.

What kinds of entities are searched or selected? When conducting searches that do not require rich relationships within data, a relational database is a better option. Questions and actions such as: Who are the people working on a project X? Sort files in ascending order by date. List organizations by nationality. These questions and actions can be satisfied with single filtering criterion or an index. While these questions can be answered with a graph database, it is not advisable to utilize graph databases in these cases. (Perryman & Bechberger 2020)

Are the relationships between entities explored? If meaning and topological value to data is being added, it is a strong indication of graph databases being the solution. Some examples of these could be: What are common subjects that Jim and Pam have been working on? How are two companies related? How does Dwight relate to a project Y? This information is leveraged by graph databases better than by any other data engines. Graph database query languages suit to reason over relationships in data. Though relational databases can answer types of questions such as friends-of-friends queries, they might require complex or difficult queries that consist of complex joins or recursion over many tables. (Perryman & Bechberger 2020)

Is data being aggregated? Relational databases are optimized for complex aggregation queries. Such questions could be: How many unfinished projects are there? What are the average sales for each day over the past two months? How many transactions are being processed each day? These questions can also be answered in a graph database, but large traversals are required which might cause higher latency or resource utilization. (Perryman & Bechberger 2020)

Are entity relations used for pattern matching? Pattern matching is a prime example of leveraging the power of graph databases. Typical examples of pattern matching are recommendation engines, fraud detection, or intrusion detection. Some example questions: Who has a similar profile as Stacy? What are some transactions that look like transaction X? Is user A the same as user B? Pattern matching is so common in graph databases that graph query languages have specific, built-in features that handle these sorts of queries. (Perryman & Bechberger 2020)

Is centrality, clustering or influence associated with the problem? A typical graph database use case is utilizing relative influence or importance of entities. Some example questions could be: Who is the most influential person whom person A is connected to? What phase in a project has the most substantial impact if delayed? What phases tend to be also delayed if phase Y is delayed? These are examples of problems in which critical pieces of some infrastructure are identified or groups of entities are located. Answers to these problems requires looking at the entities, relationships, incident relationships and adjacency. These are types of problems that often have specific built-in query language features similarly as pattern matching does. (Perryman & Bechberger 2020)

These kinds of questions help in evaluating if a problem is a good fit for solving with a graph database. Understanding suitable graph database use cases should ensure recognizing opportunities to utilize the right tool. Graph database technology is not a silver bullet for database needs, though graphs can be used to model almost anything.

5.6 Graph database suitable use cases in Enterprise Information Management

Known use cases for graph databases which suit the context of Enterprise Information Management are presented. Some of the use cases can also be implemented with relational databases, but it might require more effort. Implementation of given examples with graph databases is a good choice.

5.6.1 Master Data Management

Master data is critical to business operation. Master data includes user data, customers, products, suppliers, departments, geographies, cost-centres, sites, and business units. In large organizations, the data can reside in silos. Silos might create overlap and redundancy. Master Data Management (MDM) is the practice of identifying, storing, cleaning, and governing this data. Key concerns include change management, incorporating new sources of data, supplementing existing data with external sources of data, addressing the needs of compliance, reporting, business intelligence, and versioning data. While graph databases might not necessarily provide a full MDM solution, they are ideal to be applied to modeling, storing, and querying of hierarchies, master data metadata, and master data models. (Robinson *et al.* 2015)

Master data models include type definitions, constrains, relationships between entities, and mappings between the underlying model and source systems. Multiple redundant data sources might lead to a requirement of being capable of handling ad hoc, variable, and exceptional data structures. Graph databases being schema free allow this capability while also allowing rapid development of the master data model aligned with changing business needs. (Robinson *et al.* 2015)

5.6.2 Knowledge graphs

Knowledge graphs are a specific type of graphs with contextual understanding emphasized. These graphs contain interlinked sets of facts that describe entities, events, and their interrelations. To be able to reason about the underlying data, organizing principles must be applied. These principles give an additional layer of organizing data. This adds connected context which supports reasoning and knowledge discovery. (Barrasa *et al.* 2021)

Taxonomies can be used for creating hierarchies. Taxonomies allow support for *x is a kind of y* reasoning. A taxonomy is a classification scheme. It organizes categories in a broader narrower hierarchy. More specific things such as instances of a category are placed toward the bottom of the hierarchy. Categories and other less numerous things are placed towards the top of the hierarchy. In addition to taxonomies, ontologies can be used to create multilevel relationships. Ontology, in this context, is a classification scheme that describes the categories in a domain and the relationships between them. They are not restricted to hierarchical structures. Ontologies allow more complex types of relationships such as *part_of*, *compatible_with*, or *depends_on*. (Barrasa *et al.* 2021)

A knowledge graph's organizing principle should be chosen by the intended use. There are standards created for variety of domains which can be utilized and refined further on for specific use. A good knowledge graph is flexible, and it should be easy to maintain. It is performant and as businesses change, so should the knowledge graph. A good and suitable organizing principle allows this. (Barrasa *et al.* 2021)

Relationships are used to describe how entities interrelate. However, relationships can be used to connect data with metadata, which is a powerful combination. A globally linked view of data makes significant use cases possible. Metadata allows for catalogues of richly described datasets. (Barrasa *et al.* 2021)

5.6.3 Actioning knowledge graph

Relationships and metadata can be used to make an actioning knowledge graph. Actioning knowledge graph allows querying data such as "What customers subscribe to service X?" like knowledge graphs but can also provide confidence in answering questions related to provenance and governance of data. (Barrasa *et al.* 2021) Provenance and governance of data is important when handling sensitive data.

Popular use cases for actioning knowledge graphs are data lineage, data catalogue, impact analysis and root cause analysis, and information search. Data lineage traces all steps in data pipelines. This provides trust and high-fidelity provenance information. (Barrasa *et al.* 2021) High-fidelity provenance establishes higher security by increasing confidentiality and by allowing tracking who've accessed information and when.

Impact analysis and root cause analysis are two similar use cases. Concrete examples of these use cases are risk management, service assurance, ultimate beneficiary ownership, or fraud origination. If a company is faced with an information leak or other fraudulent activity, the actioning knowledge graph could be queried for individuals who have accessed the information. The actioning knowledge graph could detect suspicious or fraudulent activity or behavioural patterns. Such as material marked as sensitive is viewed by a non-stake holder – user who should not have rights to view, modify or download sensitive material should be detected when doing so. CSP should ensure high security as it is used to access most of an enterprise's information assets, especially if CSP should satisfy the needs of collaboration with external stakeholders.

5.6.4 Machine learning

Applying graphs in machine learning is not a new idea but only during the past few years this has gained more interest. Although many machine learning technologies rely on graphs, they neither allow graphs as input nor as output. Fixed vectors or matrices of data are accepted as input by most standard machine learning algorithms. (Perryman & Bechberger 2020)

Due to the need of using vectors and matrices for applying standard machine learning algorithms, ways of extracting graph data into vectors and matrices must be applied. A simple method for using graphs in machine learning is to extract features. There are graph analytics algorithms that can be used to extract features of a graph. When these graph features are combined to create a vector or a set of vectors for machine learning, it is called graph embedding. (Perryman & Bechberger 2020)

Sparse data is converted into more compact vector representations in graph embedding. Generally, there are two forms of embedding: node embedding and graph embedding. In node embedding, each node is represented as a single vector or matrix to compare items on a node level. In graph embedding, an entire graph or a sub-graph is represented in a single vector or matrix. Vectors and matrices are simpler and faster than comparable operations on a graph, which makes operating on those desirable. (Perryman & Bechberger 2020) Important aspect is deciding what features should be extracted from a graph.

Extracting features from a knowledge graph can be used to create a context aware AI. Context awareness improves reasoning. Instead of extracting information from a plain graph, a knowledge graph may be used for better reasoning. AI could then reason predictions and discovery of interesting data via applied ontologies and taxonomies. The data that AI creates can then be fed back to the knowledge graph, making the knowledge graph smarter. AI and knowledge graphs are driving breakthroughs and competitive advantage for organizations, but the combination is where the industry is heading. (Barrasa *et al.* 2021)

5.6.5 Other mentionable use cases

A graph database can be used to create a full text search feature. Unstructured data such as contracts, instructions, memos, and other documents which contain natural language text can be scanned for their content while gathering information such as appearance of words and counts. By using this information based on natural language text, a graph can be built in which resides all the words found in the content and the documents would be connected to adjacent words.

In addition to graph database technology providing possibilities for building features into a CSP product, it could also provide use in Business Intelligence (BI) for a CSP vendor. Analytics data could be used to create a graph. There might be patterns to detect from analytics which could provide insight into how the product is utilized. For example, when introducing new features to the software, it could be interesting to see how the utilization of features changes. How often are some features being used, do customers with whom cooperation was terminated have something in common, do customers who face issues with the product have something in common etc.?

Diagnostics could be collected to detect anomalies in the product. For instance, have some features slowed down after an update, have more errors occurred when something was changed etc.? There might be insight to be undiscovered in BI and analytics, which could provide valuable information allowing proactive and reactive actions to take place.

5.7 Summary of graph databases

Though graph database technology has been present for multiple decades, they have only recently started to gain attention. This is due to big companies such as Google, Facebook and Twitter successfully centering their business around their proprietary graph database systems and due to introduction of general purpose graph database technologies. Graph databases have not had time to mature as is the case with relational databases and some NoSQL databases. This has not stopped some of the most influential business's from successfully utilizing graph databases.

Labeled property graphs present a good general purpose graph database. It is a natural way of modeling data to label entities, give them properties and connect them by named relationships. Though it may seem simple to model data, modeling application domains requires careful consideration. Modeling data should comply to the needs of the application as this has an impact on the way querying is done and the possibilities of optimization. Selection of a graph DBMS has its own impacts also.

Though graph databases, and especially labeled property graphs, present a good general purpose database solution, they are not suitable for all databases needs. This is good to acknowledge when choosing between graph databases and other database types. Some suitable use cases were presented as well as indications of use cases which are not. Same data can be stored in both types of databases, but modeling, storing, and querying is different. These differences are demonstrated in the next Chapter.

6. COMPARISON OF RELATIONAL DATABASES AND GRAPH DATABASES

In this Chapter, relational databases are compared to graph databases. The comparison is done to highlight and argue some mentioned key characteristics. Comparisons of modeling, querying and performance are investigated. As the de facto language for relational databases, SQL is compared to Cypher. Since there is no standard query language for graph databases, Cypher is selected as the query language of Neo4j which is also recognized as the leading graph data platform by Gartner in Q4 2020 (Yuhann 2020). The source of the language comparison is Neo4j's developer documentation. It can be argued that using a graph DBMS provider's documentation would be a biased. However, the examples are simple and for example large recursive SQL-queries are not addressed, which logic have been mentioned to be significantly easier to handle with graph databases (Bechberger 2019). Performance-wise a good comparison of MySQL and Neo4j has been conducted by Vicknair *et al.* (2010) which will be referred to. Though the comparison is over a decade old, the results are still relevant. For performance, the results of a comparison which is referred to, are not generalizable. Recall for graph databases, the performance is affected by multiple factors. Also, for relational databases, there are performance differences for different DBMSs. Focus is on understanding how the performance is affected and therefore the publication is still relevant.

6.1 Comparison in modeling and querying

In this Chapter, data modeling of a relational database and a labeled property graph is compared. This comparison should emphasize the fundamental difference in data modeling. The source of the query comparison is also from Neo4j’s documentation.

6.1.1 Data models

Microsoft's publicly available sample database Northwind is used as the example. It represents a retail application's data storage. The database contains customers, products, orders, employees, shippers, categories, and their interactions. The following relational database queries refer to the model which is represented in figure 7.

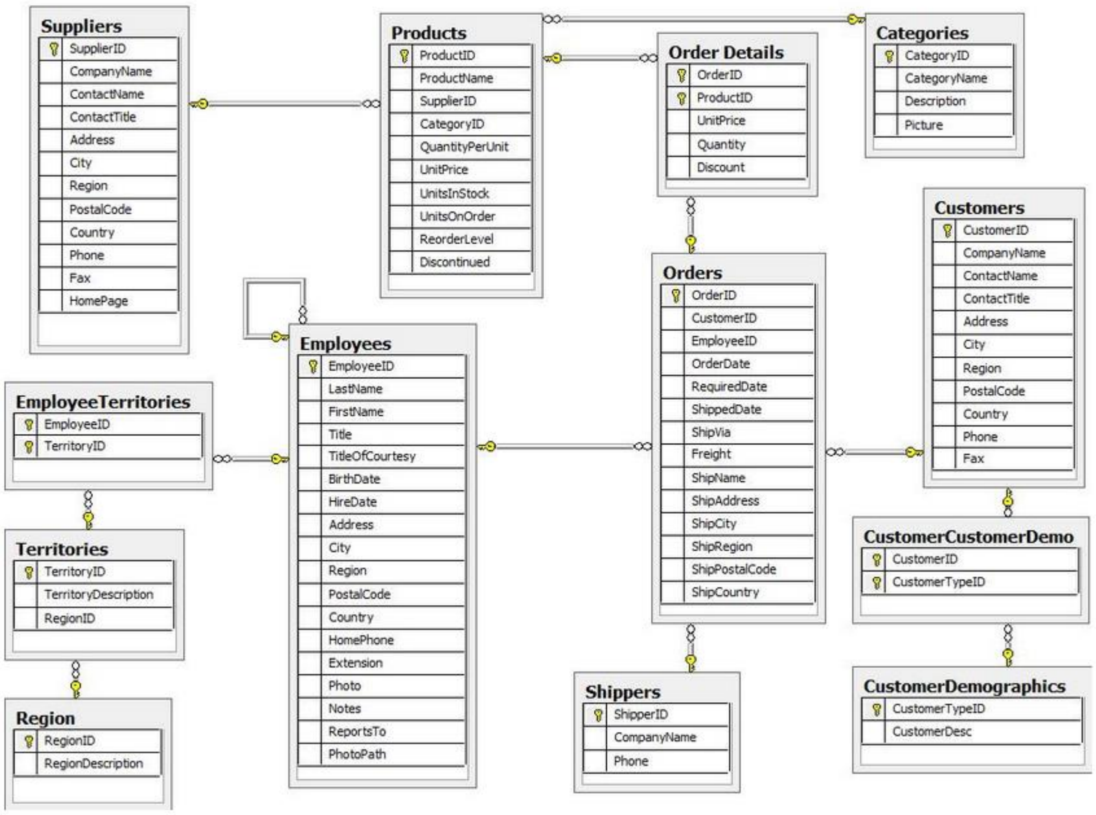


Figure 7. *Relational model of Northwind database* (Neo4j Inc.).

The graph model of the Northwind database is presented in figure 8.

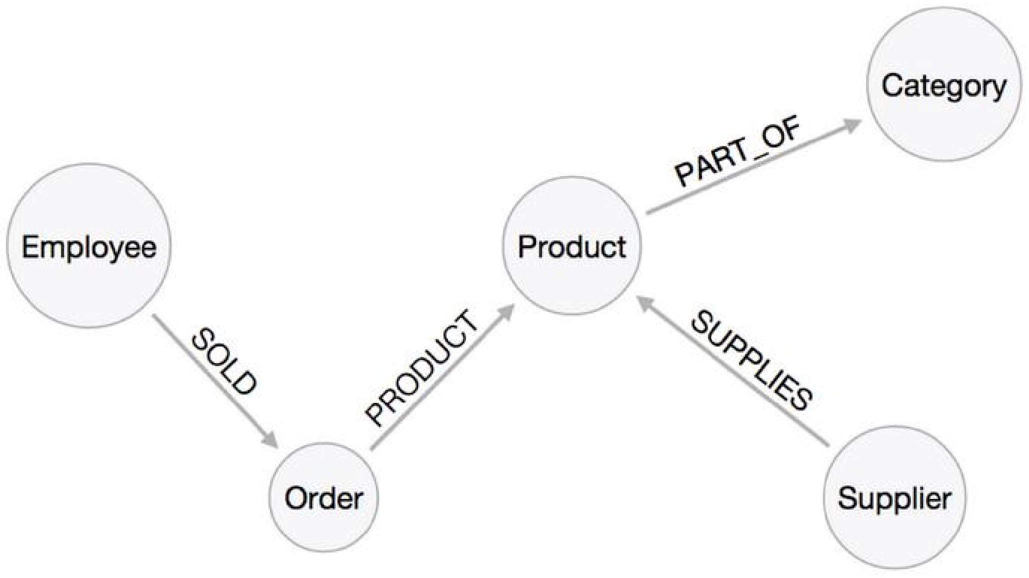


Figure 8. *Graph model of Northwind database* (Neo4j Inc.).

The graph model does not contain the node properties which makes it simplified. The graph model encapsulates a lot of information. Relationships are depicted with less edges in the graph model as there is no need to create tables for representing complicated relationships as is with *Order Details* -table. The graph model is somewhat ambiguous as it is not clear how customers are modeled.

6.1.2 Simple queries

Queries are based on the presented data models. This is not a comprehensive comparison as types of queries are missing such as recursive queries. The point is to highlight some key differences in how queries are structured.

Finding all products in SQL is a simple select from the *products* table:

```
SELECT p.*

FROM products as p;
```

Program 1. *Selecting all products in SQL* (Neo4j Inc.).

The equivalent in Cypher is a simple match pattern which selects all nodes with the label *:Product* and returns them:

```
MATCH (p:Product)

RETURN p;
```

Program 2. *A declarative pattern which matches all products in Cypher* (Neo4j Inc.).

Noticeable is how the keyword *MATCH* is used in Cypher. The essence of Cypher are statements that represent patterns. In a node pattern *(variable:Label)* a variable and one or more labels for the node can be used. Attributes can also be provided as a key-value structure e.g. *(item:Product {name:"Chocolate"}).* (Neo4j Inc.) In this example, item is the name given to the variable in the query and it represents one or more nodes labelled as "Product" and has a name attribute with the value "Chocolate".

It is more efficient to return only a subset of attributes. Next example query returns a subset of attributes, orders by price and returns only the 10 most expensive items:

```
SELECT p.ProductName, p.UnitPrice

FROM products as p

ORDER BY p.UnitPrice DESC

LIMIT 10;
```

Program 3. *SQL query of the ten most expensive products' name and price ordered descending by price* (Neo4j Inc.).

The equivalent of this in Cypher would be:

```
MATCH (p:Product)

RETURN p.productName, p.unitPrice

ORDER BY p.unitPrice DESC

LIMIT 10;
```

Program 4. *Querying ten most expensive products' name and price, ordered descending by price in Cypher* (Neo4j Inc.).

The syntax is somewhat similar. In both queries the variable used for a product is named *p* and a similar syntax is used to select the attributes, disregarding casing used for attributes. Ordering and limiting are syntax-wise similar. Similar syntax helps understanding Cypher if SQL's syntax is familiar.

Selecting a single *Product* named *Chocolate* in SQL would be done as follows:

```
SELECT p.ProductName, p.UnitPrice

FROM products as p

WHERE p.ProductName = 'Chocolade';
```

Program 5. *SQL query of the product Chocolate* (Neo4j Inc.).

The equivalent in Cypher would be:

```
MATCH (p:Product)
WHERE p.ProductName = "Chocolade"
RETURN p.productName, p.unitPrice;
```

Program 6. *Cypher query of the product Chocolade* (Neo4j Inc.).

A shortcut can be used to rewrite program 6's query by using a key-value structure:

```
MATCH (p:Product {productName:"Chocolade"})
RETURN p.productName, p.unitPrice;
```

Program 7. *Cypher query using a key-value structure to get a product named Chocolade* (Neo4j Inc.).

Queries that do not utilize relationships can be found to have similar syntax. It must be noted that the presented SQL queries could have been written without using variables in the queries. This could have made the SQL queries simpler. The variables might have been used to create analogy to the Cypher queries. This is also present in the following queries.

6.1.3 Queries utilizing relationships

The most notable difference can be perceived in how the queries are structured when relationships are utilized as can be noticed in the next queries.

```
SELECT DISTINCT c.CompanyName
FROM customers AS c
JOIN orders AS o ON (c.CustomerID = o.CustomerID)
JOIN order_details AS od ON (o.OrderID = od.OrderID)
JOIN products AS p ON (od.ProductID = p.ProductID)
WHERE p.ProductName = 'Chocolade';
```

Program 8. *SQL query for customers who bought chocolade* (Neo4j Inc.).

The equivalent of the query in Cypher is more concise as there is no need to join tables and instead a graph pattern is used.

```
MATCH   (p:Product   {productName:"Chocolade"})<-[:PRODUCT]-
(:Order)<-[PURCHASED]-(c:Customer)

RETURN distinct c.companyName;
```

Program 9. *Cypher query for customers who bought chocolade* (Neo4j Inc.).

The query is more concise, but the syntax may seem complex at first. It helps to be aware of the ASCII art -like syntax of Cypher; a graph is depicted as ASCII art. A node pattern as mentioned earlier is depicted as follows: (*variable:Label).* The query represented in Program 9 additionally uses a key-value structure. For relationships the pattern is: *()-[someRel:REL_TYPE]->().* The round brackets represent the nodes, and the square bracket labels the relationship. (Neo4j Inc.) The direction of the relationship is depicted by the direction of the arrow. The query in program 9 also implies that the presented graph model is incomplete as it refers to a node labeled as *Customer* even though it is not present in figure 8.

To know what a customer has bought and paid in total, *OUTER JOIN* must be used if there are customers without orders to be included.

```
SELECT   p.ProductName,   sum(od.UnitPrice   *   od.Quantity   )
AS Volume

FROM customers AS c

LEFT OUTER JOIN orders AS o ON (c.CustomerID = o.CustomerID)

LEFT OUTER JOIN order_details AS od ON(o.OrderID= od.OrderID)

LEFT OUTER JOIN products AS p ON (od.ProductID = p.ProductID)

WHERE c.CompanyName = 'Drachenblut Delikatessen'

GROUP BY p.ProductName

ORDER BY Volume DESC;
```

Program 10. *SQL query to find out what company Drachenblut Delikatessen has bought and paid in total* (Neo4j Inc.).

In Cypher the match becomes *OPTIONAL MATCH*, the equivalent of *OUTER JOIN* as shown in program 11.

```
MATCH (c:Customer {companyName:"Drachenblut Delikatessen"})
OPTIONAL MATCH (p:Product)<-[pu:PRODUCT]-(:Order)<-
[:PURCHASED]-(c)
RETURN p.productName, toInteger(sum(pu.unitPrice *
pu.quantity)) AS volume
ORDER BY volume DESC;
```

Program 11. *Cypher query to find out what company Drachenblut Delikatessen has bought and paid in total* (Neo4j Inc.).

Again, due to the pattern matching in Cypher, the query can be expressed more concisely. Being concise does not make it simpler and it is the ASCII art -like syntax which can make the query simpler. Cypher's syntax is aligned with the idea of graphs being natural to imagine.

So far, examples of simple queries, field accessing, ordering, paging, filtering, joining, distinct results, outer joins, and aggregation have been explored. To keep the list non-exhaustive, some examples have been omitted. Last example addresses self-joining. This is the case when expressing category-, territory- or organizational hierarchies in SQL. When getting into multi-level queries, the number of joins grow considerably. The query in program 12 explores three levels of self-joining.

```
SELECT p.ProductName

FROM Product AS p

JOIN ProductCategory pc ON (p.CategoryID = pc.CategoryID AND
     pc.CategoryName = "Dairy Products")

JOIN ProductCategory pc1 ON (p.CategoryID = pc1.CategoryID
JOIN ProductCategory pc2 ON (pc2.ParentID = pc2.CategoryID
AND pc2.CategoryName = "Dairy Products")

JOIN ProductCategory pc3 ON (p.CategoryID = pc3.CategoryID
JOIN ProductCategory pc4 ON (pc3.ParentID = pc4.CategoryID)
JOIN ProductCategory pc5 ON (pc4.ParentID = pc5.CategoryID
AND pc5.CategoryName = "Dairy Products")
```

Program 12. *SQL query expressing hierarchy depth of three levels* (Neo4j Inc.).

It can be noted from the query in program 12 that the number of opening and closing round brackets don't match. The example is taken as is. This SQL query is a bit laborious to write compared to the Cypher version, which is again more concise and instead of three levels, deals with hierarchies of any depth.

```
MATCH (p:Product)-[:CATEGORY]->(l:ProductCategory)-[:PARENT*0..]-
(:ProductCategory {name:"Dairy Products"})

RETURN p.name
```

Program 13. Cypher query expressing variable length paths (Neo4j Inc.).

Variable levels of hierarchy are represented by variable length paths. A star * is used to denote variable levels to which can also be applied optional limits *(min..max)* (Neo4j Inc.).

The purpose of these examples was to emphasize the difference in modeling, and how the querying in Cypher differs from SQL, in an understandable scope. Relational databases are based on well-defined schema and graph database models are unrestrictive as themselves. The modeling of the graph database applies to property graphs overall and the used modeling is not restricted to Neo4j. A similar graph model could be constructed in any labelled property graph.

6.2 Example MySQL and Neo4j performance comparison

This comparison refers to Vicknair *et al.* (2010) who conducted a comparison of MySQL and Neo4j performance from data provenance perspective. Rather than trying to come up with generalizable performance differences between relational databases and graph databases, the results help understand how performance is affected. The paper was selected due to the perspective being aligned with the interests of this thesis.

6.2.1 Data provenance

Provenance of data is the lineage of that data. Lineage describes what the data is and how it came to be. Provenance of a data item includes details about the processes and input data, which together were used to create the item. As mentioned in Subchapter 2.4, the source of data can vary. Complete provenance includes all the processes and versions of the item that came to be to create the item. Provenance can exist at different granularities such as from entire databases to tuples within databases, or files. (Vicknair *et al.* 2010)

It is recommended to model provenance using directed acyclic graphs (DAGs) (Vicknair *et al.* 2010; Khan *et al.* 2019). A sample DAG is presented in figure 9.

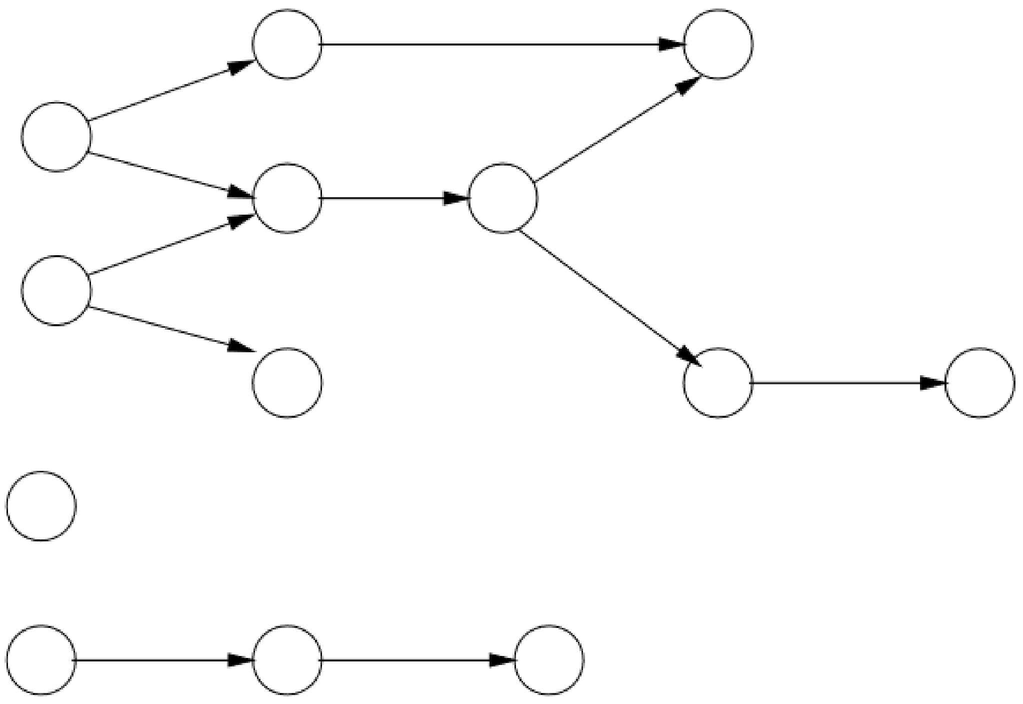


Figure 9. *A sample directed acyclic graph* (Vicknair *et al.* 2010).

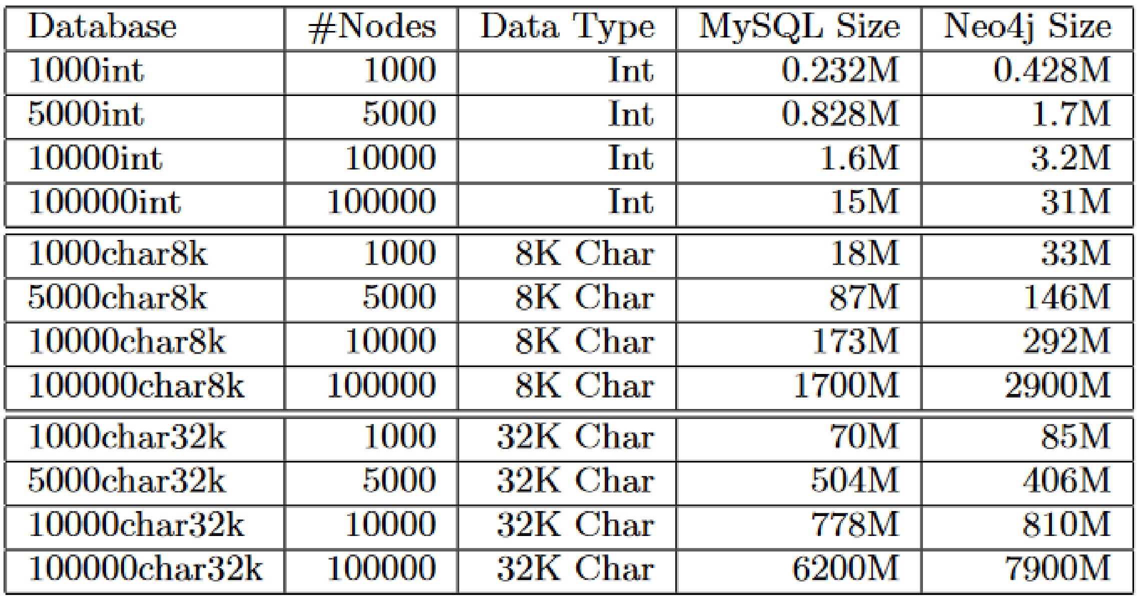
A sample use case of these acyclic graphs could be document versions. A descendent would then be a newer version of a document. The root of a cycle would be the initial version. A branching point would represent a point in which two different versions based on the same version were created. When two branches unite, a descendent is created based on the two versions. Other example of a very similar case for DAGs is the open-source version control system Git, which is a commonly used system for managing changes in software development.

6.2.2 Database construction and performance evaluation

Acyclic graph models were constructed in a total of 12 MySQL databases and 12 Neo4j databases. Each database stored a DAG consisting of some number of nodes and edges. Each node was associated with some payload which represented data that might be associated with a DAG's data point. The graph was represented in the relational database by node tables and edge tables. In the relational database each record in the node table contained a payload attribute as well as did each node in the graph database. In the relational database, a node record contained a node ID and a payload, and each edge record contained a source and a sink. Sources and sinks were foreign keys to node ids. (Vicknair *et al.* 2010)

Graphs were created with approximately 1000, 5000, 10 000 and 100 000 nodes. These number of nodes enable assessing scalability. Types of payload data also varied. Payloads were constructed to contain random integers, random 8KB strings and random 32KB strings. A table describing the databases is presented in table 1.

Table 1. *Database descriptions with disk sizes* (Vicknair *et al.* 2010).



Database	#Nodes	Data Type	MySQL Size	Neo4j Size
1000int	1000	Int	0.232M	0.428M
5000int	5000	Int	0.828M	1.7M
10000int	10000	Int	1.6M	3.2M
100000int	100000	Int	15M	31M
1000char8k	1000	8K Char	18M	33M
5000char8k	5000	8K Char	87M	146M
10000char8k	10000	8K Char	173M	292M
100000char8k	100000	8K Char	1700M	2900M
1000char32k	1000	32K Char	70M	85M
5000char32k	5000	32K Char	504M	406M
10000char32k	10000	32K Char	778M	810M
100000char32k	100000	32K Char	6200M	7900M

Inspecting table 1 and the disk sizes of the databases, noticeable is how Neo4j databases consume more space, about 1.25 to 2 times the size of the corresponding relational databases. Full text indexing was used in both the MySQL and the Neo4j databases. The DAGs structure and payload were randomly generated. The payload data consists of a series of words that were randomly selected from a dictionary that contained around 112 000 words. Data was randomly generated to each type of the 12 databases, so that corresponding Neo4j and MySQL databases were logically identical structure- and payload wise. This identical structure ensures correct comparison of the queries. (Vicknair *et al.* 2010)

The DAGs were created in layers. Two random sized sets of nodes were created representing the first two layers and random number of edges were created between the two layers. Then with similar logic, additional layers were constructed layer by layer until a graph of at least target size was created. MySQL Community Server version 5.1.42 and Neo4j version 1.0-b11 were used running on a computer with Ubuntu Linux version 9.10 which had an Intel Core 2 Duo CPU running at 3.00 GHz and 4GB of RAM. (Vicknair *et al.* 2010)

Queries simulate some types of queries that are used in provenance systems. As an example, traversals are needed to determine data objects that have been derived from some other objects or earlier revisions of that data object. Think of a linked list in the acyclic graph. If a data object is noticed to contain undesired changes, the graph is traversed to find a parent node containing a suitable basis. Another common operation is to search specific values within payloads, such as in text searches. Thus, queries were divided into structural and data queries. The structural queries refer to the structure of the DAG and data queries refer to the payloads. Three types of structural queries were defined as follows:

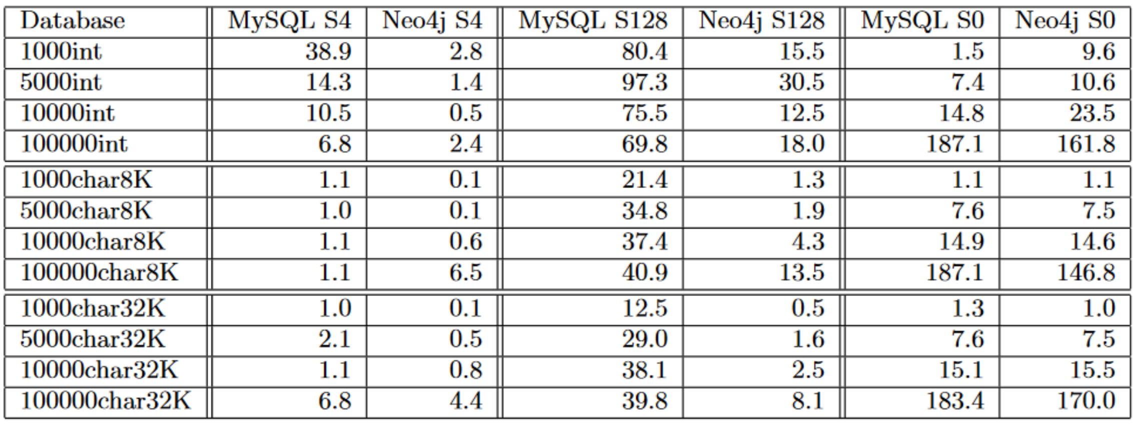
S0: Find all orphan nodes. Orphan nodes have no incoming or outgoing edges.

S4: Count the number of nodes that are reachable while traversing the graph to a depth of 4.

S128: Count the number of nodes that are reachable while traversing the graph to a depth of 128.

For each query type, the queries were run 10 times on each database while recording execution times. The longest and shortest times were excluded, and the average was counted from the remaining eight execution times. The results are presented in table 2.

Table 2. *Results of the structural queries. Execution times are presented for each type of database in milliseconds.* (Vicknair *et al.* 2010)



Database	MySQL S4	Neo4j S4	MySQL S128	Neo4j S128	MySQL S0	Neo4j S0
1000int	38.9	2.8	80.4	15.5	1.5	9.6
5000int	14.3	1.4	97.3	30.5	7.4	10.6
10000int	10.5	0.5	75.5	12.5	14.8	23.5
100000int	6.8	2.4	69.8	18.0	187.1	161.8
1000char8K	1.1	0.1	21.4	1.3	1.1	1.1
5000char8K	1.0	0.1	34.8	1.9	7.6	7.5
10000char8K	1.1	0.6	37.4	4.3	14.9	14.6
100000char8K	1.1	6.5	40.9	13.5	187.1	146.8
1000char32K	1.0	0.1	12.5	0.5	1.3	1.0
5000char32K	2.1	0.5	29.0	1.6	7.6	7.5
10000char32K	1.1	0.8	38.1	2.5	15.1	15.5
100000char32K	6.8	4.4	39.8	8.1	183.4	170.0

For these structural queries, no joins were needed to conduct in the relational database. Only the edge tables were accessed as there is no need to consider the payload. (Vicknair *et al.* 2010) Note, when counting orphan nodes with the integer databases, execution times were slower in Neo4j, even though the payload was not accessed. The same was detected in the data queries regarding integer databases. This is due to Lucene, the indexing engine used by Neo4j; it used to handle all data as strings during the making of the referred paper. However, Lucene now supports indexing of numerical data, as well as other types of data. Therefore, the comparison of integer data queries has been excluded as the results are no longer relevant.

For the structural queries, Neo4j was clearly faster. Neo4j was sometimes even 10 times faster. In the relational database, breadth-first searches were performed and a *SELECT* to the database was done for each node removed from the queue of nodes to traverse. (Vicknair *et al.* 2010) Neo4j being fast compared to the relational database is most probably due to the implementation of index free adjacency. In Neo4j, each node contains an "index" made of physical pointers. Looking up the descendent is a quick operation compared to a relational database in which the descendent is searched from a table. The table contains also other relationships, which are not relevant when searching a for a specific one. Checking of the orphan nodes in the relational database was done using the following query:

```
SELECT COUNT(*)
FROM node
WHERE node.nodeid NOT IN (SELECT source FROM edge)
AND node.nodeid NOT IN (SELECT sink FROM edge);
```

Program 14. *Counting of the orphan nodes in SQL* (Vicknair *et al.* 2010).

When processing orphan nodes, Neo4j seemed not to offer any significant benefits. However, when conducting traversals utilizing relationships, Neo4j offers significantly better results.

The full-text searches with character payload databases yielded interesting results. Four databases containing 8K character payload and four databases containing 32K character payload fields were created by selecting random strings from the dictionary earlier mentioned. For these databases, full-text searches regarding the payloads were performed. The query for performing full-text searches in MySQL databases is as follows:

```
SELECT count(*)
FROM pnode
WHERE MATCH (payload) AGAINST (SearchString);
```

Program 15. *SQL statement used for full-text searches* (Vicknair *et al.* 2010).

Lucene indexing was used for Neo4j databases. When tests were conducted on fully random data with only letters, Neo4j was clearly outperformed by MySQL. Thus, further testing was done with data that better resembled real world data, data that also contained spaces. With data that better resembled real world data, MySQL was much slower. Interesting is that MySQL performed slightly better on a small scale. Scaling upwards showed a clear shift in favor of Neo4j. Partial results for these searches are presented in table 3.

Table 3. *Query time results on character databases, in milliseconds. d = length of the search string.* (Vicknair *et al.* 2010)

Database	Rel	Neo	Rel	Neo	Rel	Neo	Rel	Neo	Rel	Neo
	d=4	d=4	d=5	d=5	d=6	d=6	d=7	d=7	d=8	d=8
1000char8k	26.6	35.3	15.0	41.6	6.4	41.6	11.1	41.6	15.6	36.3
5000char8k	135.4	41.6	131.6	41.8	112.5	36.5	126.0	33.0	91.9	41.6
10000char8k	301.6	38.4	269.0	41.5	257.8	41.5	263.1	42.6	249.9	41.5
100000char8k	3132.4	41.5	3224.1	41.5	3099.1	42.6	3077.4	41.8	2834.4	36.4
1000char32k	59.5	41.5	41.6	42.6	30.9	41.5	31.9	41.4	31.9	35.4
5000char32k	253.4	42.3	242.9	41.5	229.4	35.3	188.5	38.5	152.0	41.5
10000char32k	458.4	36.3	468.8	41.6	468.3	41.6	382.1	41.5	298.8	36.3
100000char32k	3911.3	41.4	4859.1	33.3	6234.8	37.3	4703.3	41.5	2769.6	41.5

Looking at table 3, it seems that Neo4j's performance was unaffected by scaling which was not the case with MySQL. Increasing the number of nodes had a significant effect on the query performance in MySQL. For a database used for text search engine purposes, results favor Neo4j. Results seemed interesting and raised a question on how Lucene's indexing achieves this. Shortly, Lucene achieves this by breaking down the payloads into number of terms. The terms are used to create an index where each term is associated with payloads that contain it. Therefore, it can be concluded that the variety of the selected strings found in the dictionary affects the performance of the queries in Neo4j. This explains how scaling seemed not to affect the performance and how using spaces affected the performance as spaces were probably used as separators for found terms. (The Apache Software Foundation 2006) This also emphasizes the impact that proper data modeling has.

6.2.3 Overview of the results

It is acknowledged that the used versions of Neo4j and MySQL are relatively old. Neo4j has had years to evolve from the conducted performance comparison to which is referred. One example of how Lucene and Neo4j has evolved, is the added support of numerical value indexing as well as temporal values. Neo4j has since also added schema indexes, which allows for automatic indexing of labelled nodes by their properties.

The performance findings are not generalizable for all graph databases nor relational databases. For instance, there is a distinct performance difference for *SELECT* queries between MySQL and Microsoft SQL Server (Software Testing Help). Also, relational databases can be supplemented with external indexing solutions. A good comprehensive performance comparison between Neo4j and Oracle is done in by Khan *et al.* (2019) which yielded similar results in favor of Neo4j in handling relationship rich data, even if the relational database used physical tuning for maximizing performance.

Emphasis was on how index free adjacency provides performance gains. This pinpoints the importance of acknowledging the underlying storage and processing engine when choosing between graph DBMS options. Another important aspect is that how graph databases scale. The query performance is not only affected by the modeling of the graph and the type of query, but the indexing used. A good example of suitable indexing is the Lucene's implementation of strings' indexing for text search purposes.

Different graph DBMSs offer different types of indexing options, and it is suggested to investigate how the indices were designed to be utilized. Otherwise, in the case of Neo4j, one might end up with unexpected and unnecessary disk space usage, which has little to no impact on the actual performance of the queries (Armbruster 2016). As graph databases have recently adopted mainstream popularity, it is important to acknowledge the relatively short history in utilizing the technology. It is worth investing into researching, consulting, and training regarding graph databases to ensure success in adopting graph database technologies and to successfully utilize the potential.

7. GRAPH DATABASES IN M-FILES

M-Files is a Finnish software product company which develops a Content Services Platform with the same name, M-Files (Henceforth M-Files will be referred as the product). M-Files is a Content Services Platform which is ranked as the visionary in Gartner's Magic Quadrant for CSP for the second year in a row (Woodbridge *et al.* 2021). M-Files provides Intelligent Information Management (IIM). The product focuses on what the information is rather than where it is stored. In contrast to traditional folder structured ECMs, in which it might be hard to find the right information when needed, information can be found based on what it is. Example of IIM is enhanced business automation combined with visibility and integration across systems and repositories. M-Files can connect multiple silos (as CSP should) which streamlines finding the needed content. Integration with multiple systems and digital tools, varying from electronic signatures to communication and collaboration platforms, ensures streamlined workflows. M-Files can also utilize artificial intelligence to categorize and find relevant information. AI creates intelligent metadata suggestions which further helps in finding relevant information and reduces manual labour associated with tagging information with the right metadata. (M-Files) There are use cases presented in graph database literature which indicate the suitability of graph databases for M-Files. Suitable use cases and hypothesis of possible benefits are introduced.

7.1 Recommendations based on patterns

Organizations gain competitive and operational advantage by leveraging social information. Discrete information about individuals and their relationships can be combined to facilitate collaboration, manage information, and predict behaviour. Social networks are a natural fit for graph databases. Insight can be generated into individual behaviour by understanding interactions. Interactions provide information about explicit social relations, but the relations can also be implicit. Implicit relations can be based on similar interests or behaviour patterns. (Robinson *et al.* 2015)

M-Files is used by customer organizations in varying fields, and they have people using the system. As people are in the centre of a functioning organization, social constructs in an organization can be identified. Explicit social constructs can be identified from the organization layout and from direct interaction. Implicit relations among people could be recognized from the way they use the system. An easy example of recognizing explicit social constructs would be to gather information regarding workflows (processes which are complied to when dealing with certain type of information). Patterns could be detected such as people who are usually interested in objects which are moved to a certain workflow state. These could be metadata based, such as a legal representative is interested before an agreement is signed. Patterns could also be implicit; a certain person is associated with interacting on certain types of objects when they are moved to a workflow state of interest. The interaction might simply be viewing the file instead of modifying it. Social relationships can be explored to generate insight into centrality, clustering, and influence of the organization's members via a knowledge graph. Computed centrality could determine the importance of a user's interaction upon objects completing their workflows. Identified clusters allow for creating social constructs which can be used for generating insight into behaviour. Influence of a user can be used for evaluating importance and relevancy towards objects or subjects. For example, a person who has most interacted with content regarding a certain customer might have most knowledge about them.

What benefit is gained from recognizing social constructs or behavioural data*?* Insight into behaviour can be used for generating effective recommendations. Recommendations are a prime example of generating end user value. It is valuable to identify people, products, or services that individuals or groups would be interested in. Social networks and recommendations provide key differentiating capabilities in retail, recruitment, sentiment analysis, search, and knowledge management. Working could be streamlined by recommending relevant information to users of interest. Instead of using precalculated and stale results, applications can surface end user real-time results that reflect recent changes to data by storing and querying this data in a graph database. (Robinson *et al.* 2015)

In M-Files, the effective recommendations could also mean suggesting actions for a user and recommending content. Such as notifying a user who is usually interested in the object when it is moved to a certain workflow state or suggest tagging an assignee. If an object has remained in a workflow state for an extended period, compared to the average deviation, the object might have been forgotten. With the same logic of how recommendations are made, users could be profiled. Users who often work with certain types of objects associated with certain metadata, could be profiled to have knowledge about the points of interests. When there is a need to find a person who has the most knowledge about a certain customer or certain types of businesses, answers could be provided. There are many possibilities opened by insight about the people and the information they handle.

7.2 Machine Learning

M-Files uses machine learning for various tasks such as creating smart metadata suggestions, finding similar objects, for calculating a relevancy score etc. The relevancy score is used to sort search results. A knowledge graph may support AI in metadata suggestions. Knowledge graph in combination with AI may provide new insight into data. AI may notice new patterns and trends in the data when ontologies and taxonomies are present. These patterns in combination with behavioural data could better enhance end user experience with more accurate predictions, insight, and personalization. M-Files is already utilizing a knowledge graph, but it is not based on a graph database. Basing the knowledge graph on a graph database would allow using graph algorithms.

For instance, AI could detect time related patterns such as new members of an organization. New M-Files user accounts would be recommended the type of material that other users then have often viewed in a similar position. In this case, the features to extract would be related to when the user was created and what position they are in. Then these would be compared to recognize implicit relations. Implicit relations would reveal other users that have been in a similar position. Via an actioning knowledge graph, it is possible to find material that the associated users either viewed often or paid careful attention to. If the actioning knowledge graph contained information about how long a user viewed a document compared to the text size of a document, AI could make approximations about the relevance or importance of a document for a user. When AI detects an ontology related to the onboarding process, it can be applied onto the knowledge graph. AI would store this detected ontology and then create effective recommendations for users that this ontology concerns. The ontology combined with the implicit relation can be used to reason the recommendation. "Other new users in a similar position have read". This is very similar to how online marketplaces recommend products. Another example could be a toolbox. If a person begins working on an object which has a certain workflow state and often in this similar situation uses a certain document as a reference, this would work as an effective recommendation. This certain document could be an instruction or a "legal check list", or document of any type which is often opened with the document of interest.

7.3 Other possible use cases and benefits

Some known to be good use cases which are relevant for M-Files have been presented. The use cases and benefits are general. Recommendations and social relations can be advantageous in a social media platform as well as in an online store or in a streaming service. There are also few use cases and advantages that are specific for M-Files.

Because M-Files utilizes relational databases, there is a risk of breaking database compatibility whenever developers make changes to data schema. Database upgrade functions which ensure data compliance must be written whenever changes are made to the schema. These upgrade functions must be programmed with care. This risk could be mitigated with graph databases as they do not force a database schema.

While a NoSQL database eliminates labour associated with relational database schema changes, more governance over the data is needed. Also, eliminating the need to accommodate database upgrades in the process of updating the software, could make updating faster as the database does not need to perform an upgrade. This could increase availability of the service as updates could be done faster and it would drive M-Files more towards cloud-native architecture.

This would only be possible if M-Files would shift from using relational databases entirely or if relevant data would be stored in the graph database. The former would imply drastic changes to the architecture of M-Files, but the latter is also possible. Hybrid solutions are possible since CSP's typically integrate multiple datastores.

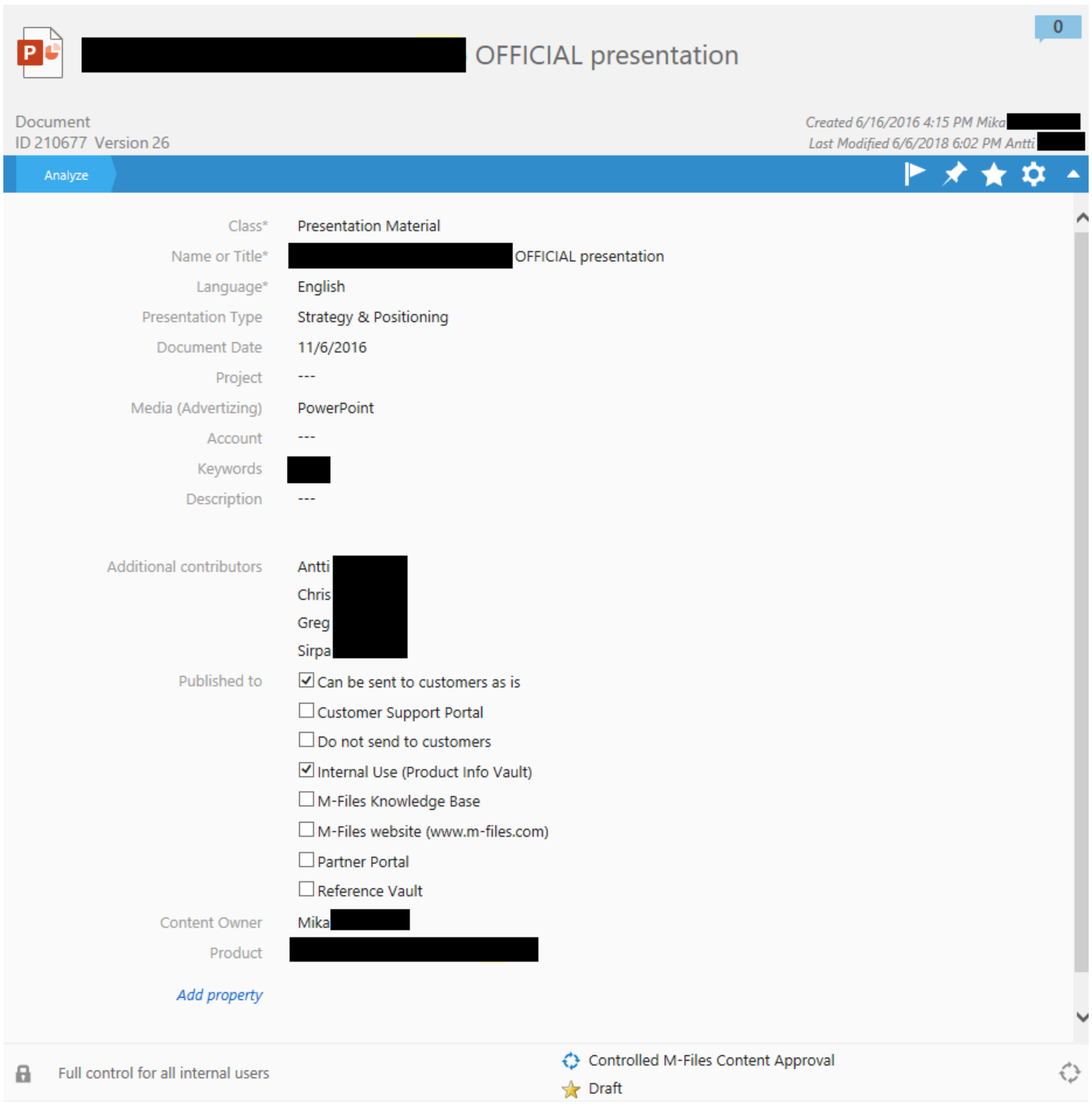
M-Files uses access control lists to determine permissions. The lists contain permissions that can be attached to objects. A list consists of one or more subjects, and operations that are either allowed or denied subject-wise. Subjects can be users, user groups, or pseudo-users. Possible operations are to delete, edit or read objects, and to change the permissions. Pseudo-users are users from metadata. An example of a pseudo-user definition would be *Project.Project manager.M-Files user*. (M-Files)

The definitions of access control can be used to create traversals in a graph database which contains the metadata. With the help of traversals, a graph can be constructed or modified to comply with permissions. Currently M-Files pre-evaluates read permissions for optimization. A graph database could provide real time access control information which quickly reflects changes.

Access control and metadata could benefit from each other in a graph. Traversals can be narrowed down by access control. Access control creates a lot of relationships in the data. This is an indication that access control could be a potential graph database use case for M-Files as this relationship-rich data could exist in the graph as an ontology.

7.4 Master Data Management

M-Files provides a solution for Master Data Management. M-Files' Intelligent Metadata Layer allows connecting to multiple different repositories to bring the information available to M-Files. Data does not need to be migrated from existing repositories as opposed to traditional Enterprise Content Management. Version history tracks data provenance. AI can be used to recognize content type and to create suggestions for metadata. This is metadata provided by Intelligent Information Management. Metadata contributes to finding the right information when needed in M-Files. An example of metadata associated with an example presentation material is depicted in figure 10.



[redacted] OFFICIAL presentation

Document
ID 210677 Version 26

Created 6/16/2016 4:15 PM Mika [redacted]
Last Modified 6/6/2018 6:02 PM Antti [redacted]

Class*	Presentation Material
Name or Title*	[redacted] OFFICIAL presentation
Language*	English
Presentation Type	Strategy & Positioning
Document Date	11/6/2016
Project	---
Media (Advertizing)	PowerPoint
Account	---
Keywords	[redacted]
Description	---
Additional contributors	Antti [redacted] Chris Greg Sirpa
Published to	☑ Can be sent to customers as is ☐ Customer Support Portal ☐ Do not send to customers ☑ Internal Use (Product Info Vault) ☐ M-Files Knowledge Base ☐ M-Files website (www.m-files.com) ☐ Partner Portal ☐ Reference Vault
Content Owner	Mika [redacted]
Product	[redacted]

Full control for all internal users

Controlled M-Files Content Approval
Draft

Figure 10. *Example metadata of a presentation shown in M-Files Desktop user interface.*

The metadata in figure 10 is shown in a user interface and is called a metadata card. The card shows associated metadata, some of which has been censored. The presentation is classified as a document, it has an ID, and the current metadata is associated with version 26. Versioning implies the presence of data governance. Other metadata includes classification as "Presentation Material", language, presentation type, date, media, keywords, who have contributed, who is the content owner and associated product. Access rights are also presented in the bottom left. In the bottom right is the name of the workflow that this material complies to. The name of the workflow is "Controlled M-Files Content Approval", and the current state is "Draft". This metadata is simple, describable, and can be used in searches. The metadata is stored in a relational database. As mentioned in Subchapter 5.6.3, metadata and relationships can be used to create an actioning knowledge graph. Metadata stored by M-Files is relationship-rich, which is not only a strong indication of a suitable use case but could contribute to other use cases also. M-Files metadata is explored in more detail in the next Chapter.

8. METADATA APPLICATION AND EFFICIENCY EVALUATION

Metadata is an important part of M-Files' functionality. Metadata is used for multiple purposes such as to categorize information, sort search results, create relationships etc. Metadata in M-Files streamlines the work experience of the end user. Categorization helps searching for specific type of information and can be used to sort search results. An M-Files user may also search for information which complies to certain conditions based on metadata. In this Chapter, the use case of storing metadata in a graph database is explored in more detail. The way M-Files stores metadata in a relational database is explained. An application which stores metadata in a labelled property graph database is presented. The data modeling of the graph database supports querying for information based on metadata constrains. The performance of executing such queries in the graph database is compared to an existing solution of M-Files. The application shows that M-Files has some readiness to use graph databases, as building the application required no changes to M-Files architecture.

8.1 Metadata in the relational database

In figure 10 was presented a metadata card of a presentation. The metadata stored in a relational database, and the file associated, compose a single version of the object. The file may be stored in another database. This object version contains metadata which can be seen in figure 10. Class, language, media, contributors etc. are called properties and their values, such as "Presentation Material", "English" and "Chris" are called property values. Depending on the definition of the property, values can be freely created, selected from a list containing created values, or they can be other objects such as users in "Additional contributors" property. The names listed are other M-Files users. An M-Files user is an object type as well. To identify a specific object version for querying the right metadata in the relational database, the primary key needs to be constructed.

The primary key of an object version is a combination of the object type, id, and object version. Then, to query the associated property values, each property value record in the database associated needs to be identified. The primary key of a property value associated with an object version is the combination of the object type, id, version, property definition and a sort-index in the case of multiselects. Multiselect means that there are multiple selectable values for the property, such as in the case of the contributors.

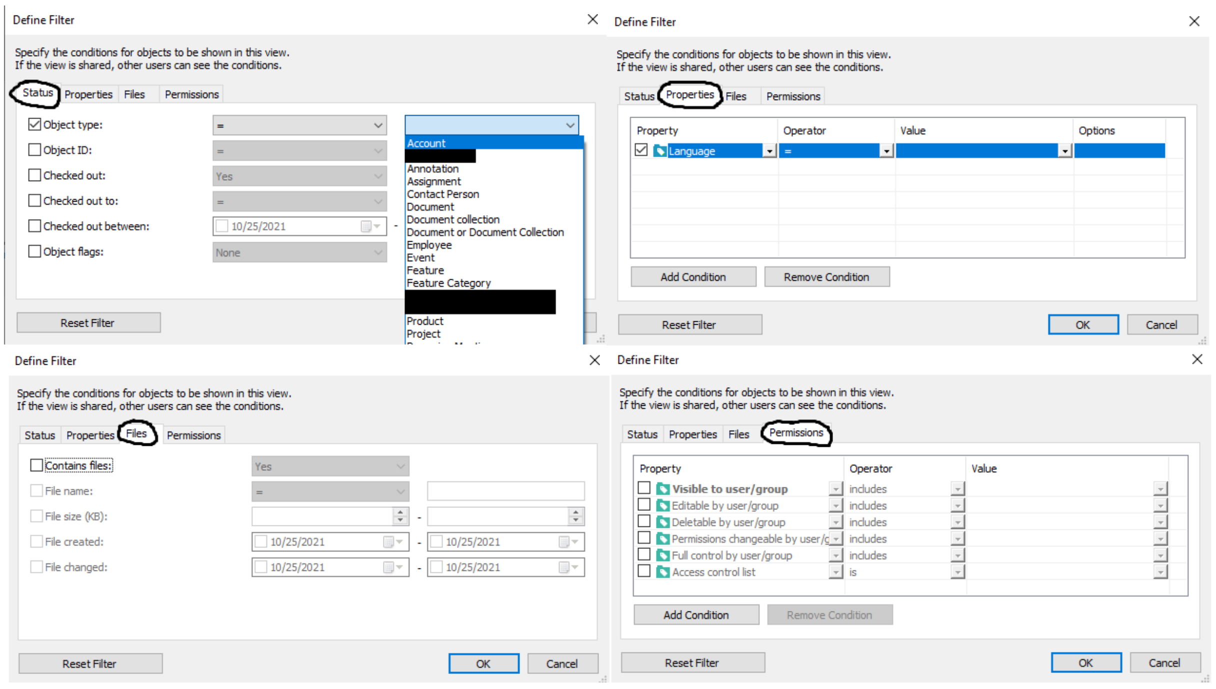
The file (PowerPoint) associated might have multiple versions as well and might be partitioned into multiple parts due to limitations of storing binary objects in the database (when the file is stored in the same database). Multiple joins might need to be conducted to create a complete object.

Access control dictates who have rights to read, modify, and delete objects and properties. In addition, access control also dictates who have the rights to change the access control. Access control lists (ACLs) contain this information and different object versions can have different access control lists.

The primary keys composing of multiple values, and probability of multiple versions of objects with many properties existing indicates large tables or multiple tables must be scanned for values. Either a huge table exists, or multiple tables exist depending on the configuration of M-Files. These tables also contain null values as columns exist for each possible datatype. Only a single attribute with the right datatype contains a value. Joins must be made to collect all the property values associated with an object. A graph database would allow connecting the value to the object version and there wouldn't be a need to scan tables to find the values. This is of course only true when there is a reference to the object. The object must be located first. Then index free adjacency would efficiently provide the associated property values without scanning unrelated data. Graph database would allow dynamically storing various datatypes and would eliminate the need to store redundant null values.

8.2 M-Files Views

A common use case in M-Files is configuring Views. "Views are locations in which the documents and other objects are listed based on the metadata they contain." (M-Files) Users can configure metadata related filters as seen in figure 11.


Define Filter

Specify the conditions for objects to be shown in this view.
If the view is shared, other users can see the conditions.

Status | Properties | Files | Permissions

- [✓] Object type: = | Account
- [ ] Object ID: =
- [ ] Checked out: Yes
- [ ] Checked out to: =
- [ ] Checked out between: 10/25/2021 -
- [ ] Object flags: None

Account
[redacted]
Annotation
Assignment
Contact Person
Document
Document collection
Document or Document Collection
Employee
Event
Feature
Feature Category
[redacted]
Product
Project

Reset Filter

Define Filter

Specify the conditions for objects to be shown in this view.
If the view is shared, other users can see the conditions.

Status | Properties | Files | Permissions

Property	Operator	Value	Options
[✓] Language	=		

Add Condition | Remove Condition

Reset Filter | OK | Cancel

Define Filter

Specify the conditions for objects to be shown in this view.
If the view is shared, other users can see the conditions.

Status | Properties | Files | Permissions

- [ ] Contains files: Yes
- [ ] File name: =
- [ ] File size (KB): -
- [ ] File created: 10/25/2021 - 10/25/2021
- [ ] File changed: 10/25/2021 - 10/25/2021

Reset Filter | OK | Cancel

Define Filter

Specify the conditions for objects to be shown in this view.
If the view is shared, other users can see the conditions.

Status | Properties | Files | Permissions

Property	Operator	Value
[ ] **Visible to user/group**	includes	
[ ] Editable by user/group	includes	
[ ] Deletable by user/group	includes	
[ ] Permissions changeable by user/g	includes	
[ ] Full control by user/group	includes	
[ ] Access control list	is	

Add Condition | Remove Condition

Reset Filter | OK | Cancel

Figure 11. *Filtering options for Views in M-Files.*

Figure 11 is a compilation of filtering option tabs. Status tab is opened in the top left. In this tab, the menu showing object type values is expanded. An object type can be selected so that the View shows only objects of the type. Examples of object types are Documents, Employees, Products and Projects.

Top right of figure 11 shows the Properties tab. Users can select multiple metadata properties for constraining the objects that are shown in the View. For instance, language property has been chosen in the figure. Different operators can be used to determine if objects of certain language should be included or excluded. Value is not selected, but by opening the Value drop down menu, the possible values for language are shown. Multiple types of properties can be selected.

Bottom left of figure 11 shows the Files tab. Users can add conditions on whether the objects should contain files, constrains on the file name, size, and dates. The bottom right shows the Permissions tab. In this tab, constrains can be applied to the View to include objects which comply to desired permissions.

The presented filtering options can be used to create Views. Views can be a powerful tool used to streamline finding the right data. Finding right information when needed saves time and increases efficiency. For the views to be effective, the objects should have accurate metadata attached to them.

8.3 Neo4j experiment

An experiment was conducted with Neo4j. This labelled property graph database was selected for proof-of-concept -use as Neo4j sandbox offers a quick local deployment and suits the needs of the database for this use case. An application was made which populated the graph database with the latest object versions metadata. The application logic is presented in figure 12.

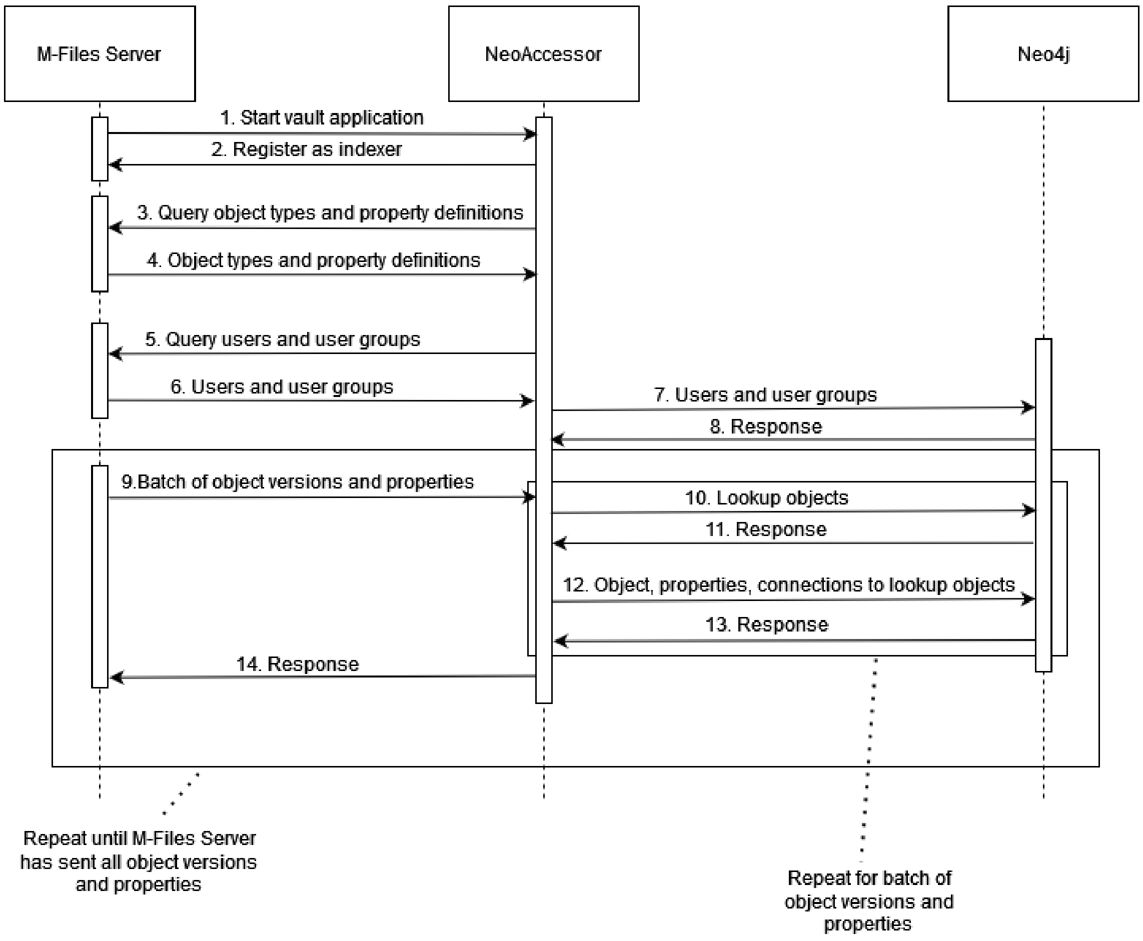

M-Files Server
NeoAccessor
Neo4j
1. Start vault application
2. Register as indexer
3. Query object types and property definitions
4. Object types and property definitions
5. Query users and user groups
6. Users and user groups
7. Users and user groups
8. Response
9.Batch of object versions and properties
10. Lookup objects
11. Response
12. Object, properties, connections to lookup objects
13. Response
14. Response
Repeat until M-Files Server has sent all object versions and properties
Repeat for batch of object versions and properties

Figure 12. *Experimental graph database application sequence chart*.

In centre of figure 12 is *NeoAccessor*, which is the name of the experimental application. *NeoAccessor* populates a Neo4j database with M-Files objects' metadata. The application inherits *IObjectIndexer* interface from *MFiles.Extensibility* framework and uses M-Files API to communicate with the M-Files Server. The application is first started by the server. Then the application registers as an indexer. After registration, the application begins by querying object types and property definitions, which are cached. Caching speeds up the process of creating Cypher queries, which are sent to the Neo4j database. Then, users and user groups are queried and sent to the database. Next, the application is ready to receive objects from the M-Files server. In the graph database, each object is stored as a node with an ID. All the property values of the object are stored as separate nodes and are connected to the object node. Relationships are given labels that indicate the property definition. The property value nodes are also connected to property definition nodes. This ontology of values connected to property definition nodes enables querying values of a property definition and the adjacent objects. Connections between property definitions, property values and objects allow for querying objects with metadata constrains.

The application is an M-Files Vault application (also known as an external application) written in C#. The application is installed for use in an M-Files Vault. M-Files Vault is a single centralized storage location for documents and other objects, which is physically located on the server running M-Files Server (M-Files). A local server was setup for these experiments, and it is based on a backup of real data. The Vault running on the server is a backup of a Vault used internally at M-Files Corporation. The Vault contained around 650 000 objects. There were over 28 different types of objects which could be classified in over 100 different ways. An agreement, invoice or email may all represent a document object type but represent different types of classification.

The Vault application was installed on the Vault using M-Files Admin, which is an application used for administrative operations. In the configuration panel created for this application, a special purpose button was created, which resets the state of the indexer. This is because the application was designed to populate the database with a single run. Otherwise, the interface always receives the latest object version of an object which was created or modified. After the reset, all the objects' newest versions metadata are sent to registered indexers. The application receives the objects which are then used to create a graph in Neo4j database. A simple graph model was used to populate the database. The model is represented in figure 13.

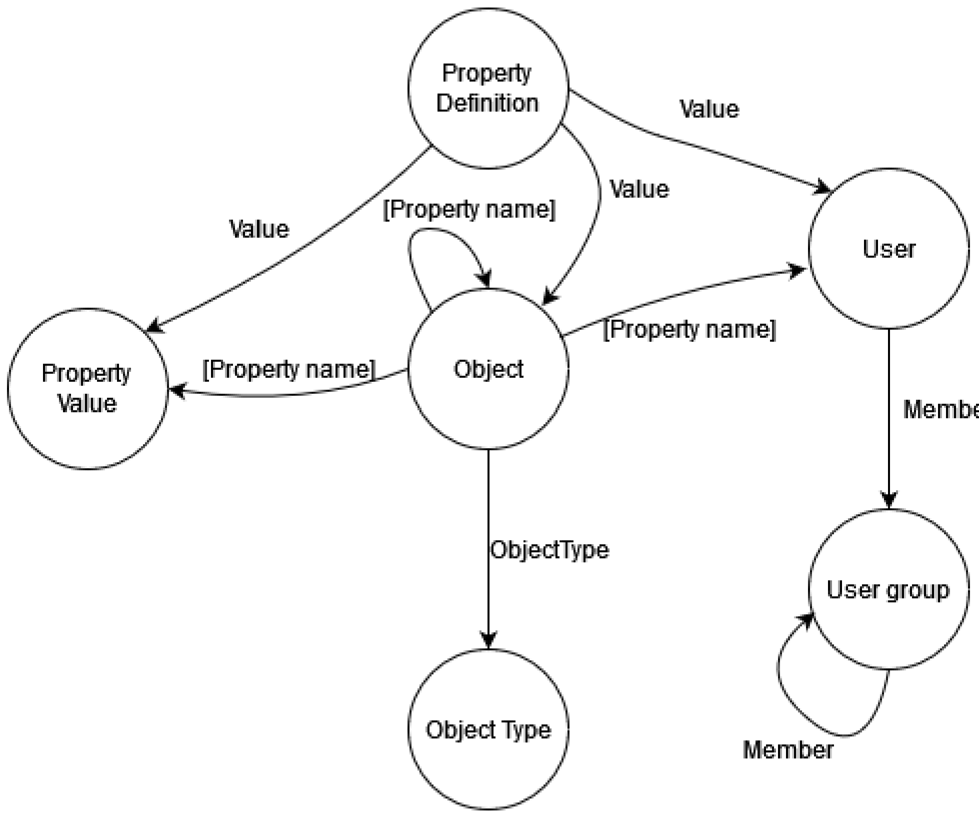


Figure 13. *Graph model of the experimental graph database.*

The graph model was designed to support searching for objects with metadata constrains like M-Files Views are used. Note that the model lacks using permissions as search filters. Users and user groups could have also been modelled as objects. However, certain built-in objects are not received for indexing via the inherited interface. It was easier to model users and user groups separately and it creates distinction between these object types.

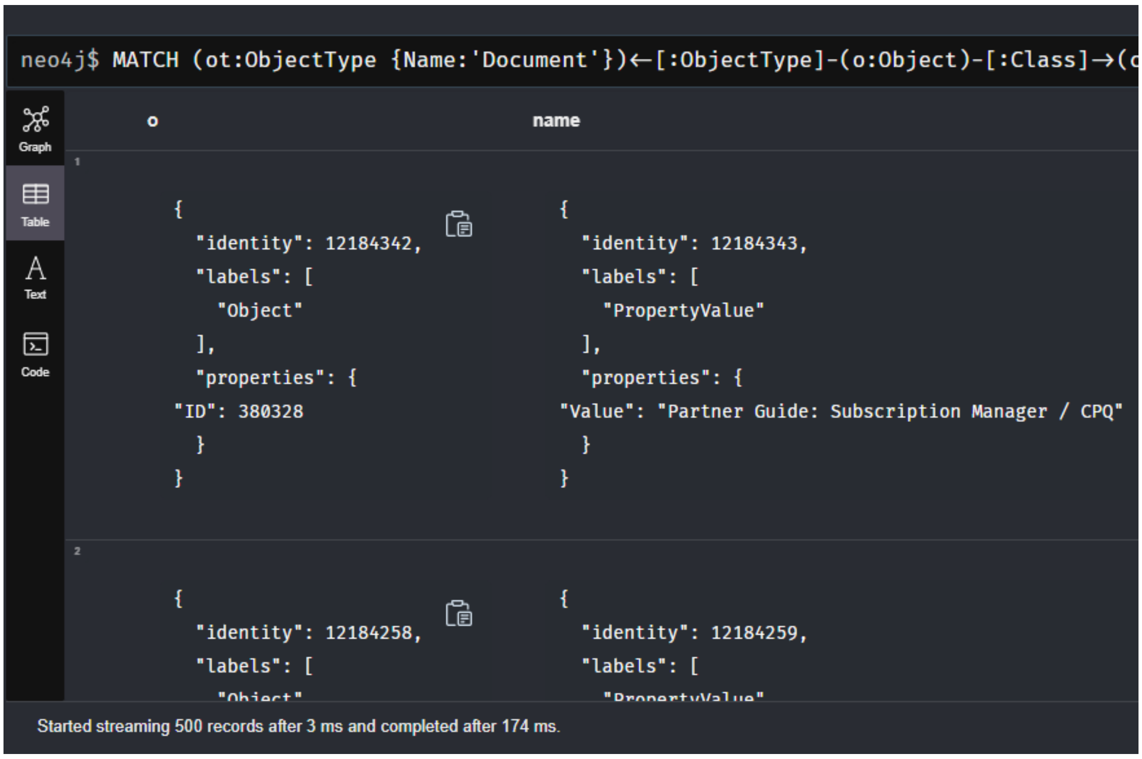
The model allows for building queries in different ways. For example, if an object type is used as a filter, a node representing the object type contains the connections to all the objects that satisfy a condition. The direction of the relationship only affects how the query is written, not performance. More filters can be applied to the already narrowed results, such as property values. Cypher queries were verified to produce valid results by comparing the graph database search results to searches made in M-Files using Views. Some example queries are presented in next comparison.

8.4 Comparison of metadata search performance between Neo4j and MSSQL

NeoAccessor populated a Neo4j database with the Vault's objects and related metadata. The graph database then contained 12,184,484 nodes and 30,790,962 relationships. In this Subchapter, Cypher query performance in the Neo4j database is compared to query execution performance in the MSSQL server used by the Vault. Multiple types of configurations can be used with M-Files. This Vault uses MSSQL, and it is compared to the populated Neo4j database.

The Cypher queries were written manually, and the SQL queries were automatically constructed by the M-Files Server. Five different M-Files Views were created. By using the same constrains, Cypher queries were created which yielded same results. Execution on the MSSQL database was logged using Microsoft Tracelog. The amount of time to get the results of a View into the listing of the user interface was logged. A listing page shows maximum of 500 results by default. All results that comply to the constrains are found, but the logged operation length is the amount of time taken to get only the results which are shown in the listing. Thus, the Cypher queries are limited to 500 results when there were more results. Neo4j's query profiler is used to get an estimate on the cost of a query. DbHits are abstract units of storage engine work (Neo4j Inc.). DbHits allows comparing the cost of Cypher queries. Although the execution times of queries vary, the profiler always gives the same estimate on the cost of a query in DbHits. The computer used for running these databases is a HP Z6 G4 workstation equipped with an Intel Xeon Silver 4114 CPU running at 2.2GHz with 10 cores and 20 threads. The computer is equipped with 32 GB RAM.

The Event Trace Log files created by Microsoft Tracelog were imported to an MSSQL database using an inhouse tool which has been created to help inspecting M-Files related database operation logs. Cypher queries were run in Neo4j sandbox. Neo4j sandbox is a desktop application which allows an easy setup of a Neo4j database and experimentation. The interface shows query execution times as seen in the bottom of figure 14.



```
neo4j$ MATCH (ot:ObjectType {Name:'Document'})←[:ObjectType]-(o:Object)-[:Class]→(c
```

```
{
  "identity": 12184342,
  "labels": [
    "Object"
  ],
  "properties": {
"ID": 380328
  }
}
```

```
{
  "identity": 12184343,
  "labels": [
    "PropertyValue"
  ],
  "properties": {
"Value": "Partner Guide: Subscription Manager / CPQ"
  }
}
```

```
{
  "identity": 12184258,
  "labels": [
    "Object"
```

```
{
  "identity": 12184259,
  "labels": [
    "PropertyValue"
```

Started streaming 500 records after 3 ms and completed after 174 ms.

Figure 14. *Snippet from Neo4j sandbox interface.*

The bottom of the figure indicates that streaming of 500 records began after 3 milliseconds, and it was completed after 174 milliseconds. The top of the snippet shows the query partially. The left tab allows for selecting different ways in how the results are shown. Table and Code selections display the query performance. Table was more suitable for this context due to clear presentation of the results. Code presented the response in a JSON string including information irrelevant for this experimentation.

Total of five different M-Files Views and Cypher queries that produced equivalent results were created. Each query was executed 10 times and the execution times were recorded. Each View was opened and recreated 10 times. Recreation was done as otherwise the results would have been fetched from a cache. The time it took to load the results into the listing from the MSSQL database was logged.

The first View was defined to gather all the accounts in Tampere city. The execution times are presented in table 4.

Table 4. *Execution times in milliseconds for getting the Tampere city accounts.*

	Accounts Tampere	
	MSSQL	**Neo4j**
	146	890
	171	839
	180	756
	189	844
	201	789
	156	789
	205	856
	144	918
	201	793
	182	888
AVG	177.5	836.2

The SQL queries are not presented as they are generated by the M-Files server. The queries are not understandable without extensive knowledge on the structure of the database and the structure is confidential information. The Cypher query is presented in Program 16.

```
MATCH ( pd:PropertyDefinition { Name: 'City' } )-[ :Value ]->( pv:PropertyValue
{Value: 'Tampere' } )<-[ :City ]-( object:Object )-[ :ObjectType ]->( ot:ObjectType {
Name: 'Account' } ) WITH object MATCH ( object )-[ :`Name or Title` ]->( name )
RETURN object,name LIMIT 500
```

Program 16. *Cypher query for getting Tampere accounts.*

The query in Program 16 follows logic in which from the node labelled as PropertyDefinition with the name City, a traversal is made to nodes labelled as PropertyValues which have Tampere as their value. From these PropertyValue nodes a traversal is made to Object nodes which have a City relationship to these nodes. From these objects are selected the nodes which have Account as their object type. The object nodes and nodes to which the object nodes have a relationship labelled as "Name or Title" are returned.

The execution times in table 4 favor MSSQL. Neo4j took almost 5 times more time to process. By using the query profiler, the problematic part of the query seemed to be in which the objects are selected which have a relationship labelled as City to the Property

Value nodes with Tampere as their value. The query profiler estimated this part of the query to cost 1667 DbHits. The actual cost of the part was 5,037,013 DbHits. Total cost of the query was 5,669,000 DbHits. The problematic part impacted the query performance most. For inspection, the object type connection was removed from the query to create a query in which the all the objects were fetched with City-relationship to Tampere. This resulting query cost total of 668,545 DbHits although the same information is fetched as in the problematic part of the original query. More investigation would be required to understand what might have caused this. The query profiler hints that this might have been affected by automatic optimization of Neo4j as the order of traversals in the query did not match how the query was written. This was confirmed by rewriting the query differently, similar results were produced.

The second View gets all objects classified as Training material and which are in Finnish language. The execution times presented in table 5.

Table 5. *Execution times in milliseconds for getting training material which are in Finnish.*

	Finnish training material	
	MSSQL	**Neo4j**
	336	124
	350	91
	392	106
	352	98
	440	81
	370	72
	470	71
	329	110
	424	99
	337	78
AVG	380	93

The Cypher query for getting the same results presented in Program 17:

```
MATCH ( ot:ObjectType { Name: 'Document' })<-[ :ObjectType ]-( o:Object )-[
:Class ]->( c:Object { ID:58 }) WITH o MATCH ( name )<-[ :`Name or Title` ]-( o )-
[ :Language ]->( lan:Object { ID:2 }) RETURN o, name LIMIT 500
```

Program 17. *Cypher query to get all the training material which are in Finnish.*

The Cypher query in Program 17 took 25 % of the time than the equivalent in MSSQL when comparing the averages of table 5. Total cost of the query was 562,639 DbHits, tenth of the cost of the previous query. A flaw in the NeoAccessor can be noticed when inspecting the query. As the *IObjectIndexer* does not index Classes or other "value lists" as was the case with users and user groups, it would be better to query all the classes from the M-Files server and send them to the Neo4j database. This would allow to utilize classes in queries in a readable format. The classes exist in the Neo4j database because all the metadata that the objects have are sent to the database and these classes are part of some objects' metadata. Still, the class objects could be used as parts of the query, because M-Files Admin lets the administrator to inspect the ID values of value list items. This is how it was known that an object to which a language relationship exists with the ID 2 represents the Finnish language.

The third View contained more metadata constrains than the previous two Views. The View lists all the objects that are in German, are classified as "Case Studies" and have "Use Case" as "Quality Management". The execution times yield interesting results.

Table 6. *Execution times in milliseconds for getting Case Studies on Quality Management in German.*

	German, Case Studies, Quality Management	
	MSSQL	**Neo4j**
	384	3
	154	3
	59	2
	90	3
	121	4
	232	3
	85	3
	92	3
	60	26
	62	3
AVG	133.9	5.3

The query returned 8 results. There is notable variance in the execution performance for MSSQL. The reason for the variance is unknown. However, even by comparing the best

MSSQL query performance of 59 milliseconds to median Neo4j performance of 3 milliseconds, Neo4j shows notable advantage. The query is presented in program 18:

```
MATCH ( ot:ObjectType { Name: 'Document' } )<-[ :ObjectType ]-( o:Object )-[ :Class ]->( c:Object {ID: 101} ) WITH o MATCH ( o )-[ :`Use Case` ]->( uc:Object { ID: 5 }) WITH o MATCH (o)-[ :Language ]->( l:Object { ID: 26 } ) RETURN o
```

Program 18. *Cypher query to get all Case studies on Quality Management written in German.*

In the query, class with ID 101 represents case studies, use case ID 5 represents quality management and language ID 26 represents German. The query cost 10,868 DbHits, notably lower than the previous query. As it was stated earlier, the performance is affected by the size of the graph covered in the query. It seems that this query resulted in a very limited size of the graph traversed.

The fourth View returned documents classified as instructions and which language was English. The execution times are presented in table 7.

Table 7. *Execution times in milliseconds for getting instruction documents which were in English.*

	English Instruction Documents	
	MSSQL	**Neo4j**
	396	149
	399	70
	290	65
	358	65
	319	64
	363	61
	371	65
	351	113
	361	75
	349	80
AVG	355.7	80.7

The performance favors Neo4j. Getting the results in Neo4j took on average fifth the time it took for MSSQL. The query is presented in program 19:

```
MATCH ( ot:ObjectType { Name: 'Document' } )<-[ :ObjectType ]-( o:Object )-[
:Class ]->( c:Object {ID:27 } ) WITH o MATCH ( o )-[ :Language ]->( uc:Object {
ID: 2 } ) RETURN o LIMIT 500
```

Program 19. *Cypher query to get instruction documents written in German.*

In the query, class ID 27 denotes classification as an instruction and language ID 2 denotes English. The total cost of the query was 449,909 DbHits. The query itself is very similar to query 2 contextually and by performance. In both, a language and classification are defined.

The fifth View was defined to get Wire transfer invoices in which M-Files Corporation was the buyer and the currency was euros. The query performance is presented in table 8:

Table 8. *Execution times in milliseconds for getting wire transfer invoices in which M-Files Corporation was the buyer and the used currency Euros.*

	Invoices M-Files Corporation EUR Wire transfer	
	MSSQL	**Neo4j**
	403	474
	456	310
	419	392
	294	344
	354	375
	375	354
	344	294
	392	419
	310	456
	474	403
AVG	382.1	382.1

The performance is tied by average. The query is presented in program 20:

MATCH (ot:ObjectType { Name: 'Document' })<-[:ObjectType]-(o:Object)-[:Class]->(c:Object { ID: 9 }) **WITH** o MATCH (o)-[:Buyer]->(uc:Object { ID: 3285 }) **WITH** o MATCH (o)-[:Currency]->(l:Object { ID:1 }) **WITH** o MATCH (o)-[:`Name or Title`]->(p) **RETURN** o,p **LIMIT** 500

Program 20. *Cypher query to get wire transfer invoices in which M-Files Corporation was the buyer and the currency was euros.*

Class ID 9 denotes an invoice, buyer ID 3285 denotes M-Files corporation and currency ID 1 denotes euros. The cost of the query according to the query profiler is 931,109 DbHits.

By dividing all the estimated query costs by the execution times, varying values can be calculated for correlation between estimated cost and actual performance.

Table 9. *Estimated costs for each query, actual performance and estimated cost divided by actual performance. Average performance is also presented to provide easy comparison of averages.*

Query	DbHits	DbHits/ms	AVG performance (ms)	AVG performance in MSSQL (ms)
1	5,669,000	6779.5	836.2	177.5
5	931,109	2436.8	382.1	382.1
2	562,639	6049.9	93	380
4	449,909	5575.1	80.7	355.7
3	10,868	2050.6	5.3	133.9

From table 9, it can be concluded that estimated DbHits are a bit inaccurate as the estimation divided by average performance yields varying results. However, this is based on a very small sample set. Overall, more estimated DbHits means longer execution. The units can be used for coarse comparison of costs.

8.5 Summary of the experiment

For the use case of M-Files Views, Neo4j could offer a viable option. The benefit of using Neo4j is uncertain as these tests were limited. The objects and the associated metadata can be stored and easily queried in a Neo4j database or any other labelled property graph. The performance was mixed but overall favored Neo4j. For some queries, the executions were clearly faster in Neo4j than in MSSQL while for some, the difference was marginal. By profiling the first query, it was noticed that the Neo4j database optimizes the order of how the graph was traversed, which in this case seemed to deteriorate performance.

Note that the performance was not the priority in the experiment whilst being an important factor. To get better information on performance, more queries and other use cases should be tested. Also, the causes of surprisingly slow query performance should be investigated and how this could be improved. Investigating this could provide valuable information which could help in evaluating Neo4j viability. Additionally, it would be worth to invest resources in training or material on graph databases to ensure valid results on the performance. The way data is modeled and queried affects the outcome when measuring performance.

The point of the experimentation was to explore graph database viability of storing metadata. M-Files provides an API which can be used to build graph database applications. There is also another, native component with non-public API, which was not utilized in this experiment, which is more performant and could have been used for the same purpose but could have only made populating the graph database faster. The comparison takes into consideration only a single use case but showed that M-Files has some readiness to adopt graph databases.

9. DISCUSSION AND CONCLUSIONS

In this thesis it was explored what graph database technology is and how they fit in the context of Content Services Platforms (CSPs). This was done to explore whether there would be benefits for M-Files from utilizing the technology. The context of CSP was established with literature on Enterprise Information Management (EIM). There is a need to integrate information in business operations to which CSPs provide a solution. CSPs evolved from the discipline of Enterprise Content Management (ECM), which failed to deliver as such. This paradigm shift from ECM to CSP is driven by digital transformation. Enterprises adapt new Enterprise Application Software (EAS) as information management needs evolve. The development and adaption of these software has been enabled by advances in information technology.

The databases used by CSPs (classified as the leaders by Gartner) was explored by varying online material which provide information about the status of databases in the context. While these materials do not reveal the whole truth as accurate architectural information is not public, they reveal that relational databases are widely used. This is not surprising as relational databases are the most used type of databases overall. The role of the used databases by each CSP remains unclear as more information on this was not searched but it was not the main point of the thesis. However, it did come across that Microsoft, the leading CSP, utilizes graph databases and sees it as a significant opportunity. Microsoft calls the technology their most important bet. It is also possible, that some leading CSPs offer additional features to their customers which utilize other databases than the primary storage. Therefore, it cannot be concluded whether these presented graph database use cases could make it possible for a CSP to differentiate from competition.

Fundamentals of relational databases and graph databases were explained with the latter in more detail by referring to literature as emphasis was on understanding graph databases. The querying of SQL and a graph database query language Cypher (used by Neo4j) was compared by utilizing ready online material found from Neo4j website. The material was found unbiased as the given examples were simple and emphasized the differences in the way of how queries are formed. A bit older (2010) but relevant publication on comparing a relational database with a Neo4j graph database was referred to emphasize the factors that influence the performance of the databases. This is still relevant as graph databases performance is affected by the amount of graph traversed in a query rather than by the size of the database as was shown. This is true for a graph database which implements index free adjacency. Index-free adjacency means that each node is directly linked to its neighbor node. In a database engine utilizing index-free adjacency, each node acts as an index of other nearby nodes. This is much cheaper than using global indexes. This is important to acknowledge as the performance gain associated with utilizing graph databases is dependent on the underlying implementation, not all graph databases implement this. But how does everything done in this thesis come together?

General conclusion from multiple sources is that graph databases are good with relationship rich data. Graph databases can be used to add meaning and topological value to data. Pattern matching, centrality, clustering, and influence of entities within data is so common in graph databases that graph query languages have specific, built-in features that handle these sorts of queries. The subject in which graph databases excel could contribute to bringing end user value and personalized experience in EIM in the context of CSPs. CSPs are amid EAS and potentially have access to the data needed to populate a graph database with leverageable data. This would allow to take advantage of graph database use cases. It was stated that metadata and graph databases are a powerful combination. End user value and personalized experience would be delivered by utilizing metadata and user action data. Detected patterns could provide insight into how the users utilize information assets and could support their ways of working as well as provide answers to problems related to metadata. Examples of end user value and personalized experience would be calculating relevancy scores on metadata for each user, which could be used to sort information. Patterns could be used to suggest related material when working or viewing some information. Documents could be recommended which are often used as help or reference when working on some specific type of information. Questions such as who has the most experience on a certain customer or topic could be answered. The examples were presented in Chapter 7.

What makes this possibly interesting for M-Files, is that the metadata needed to build these kinds of features is already stored by M-Files. The metadata is relationship rich which is a strong indication that graph databases might be a good choice as was stated in the literature review. The proof-of-concept application presented in Chapter 8 shows that M-Files has some readiness to send this metadata to a graph database. It is said that *some* as it cannot be claimed that M-Files has full readiness. Most likely some tweaks should be made such as storing detailed user action data. In addition, there are also other use cases and benefits for M-Files that are not directly associated with end user value and personalized experience. Graph databases do not force a database level schema. Therefore, there is no need to run database upgrades whenever changes are made to how the data is stored. This aspect is aligned with an idea of software complying to cloud-native architecture, something that may interest M-Files. The availability of the service could be increased. In addition, M-Files access control is by nature relationship rich. Users, user groups and objects are assigned access rights which creates relationships. Metadata searches and access control could complement each other, as these together make it possible to limit the amount of graph traversed in queries. This affects the performance of a graph database utilizing index free adjacency. Literature also states that breakthroughs in machine learning are made with the combination of graph databases, hinting more future innovations on the technology.

The presented use cases and possible benefits can come across as **bold** statements, much is promised. However, the literature states that these are common types of problems that are solved with graph databases. Building these features into a CSP is possible (especially for M-Files) and recommended to be done with graph databases. It is not claimed whether it is not possible to achieve this with relational databases, but graph databases are better in handling relationship rich data and solving related problems, a tool for the job. The unanswered question can only be answered by leaders at M-Files who make the decisions on prioritization and technology, as they have knowledge and understanding of what customers appreciate, whether the value is worth the effort? More information and experimentation may be desired to make decisions. There is no need to go all-in on graph databases to leverage some of the benefits. Literature suggests on beginning to work on a single use case and expanding to other use cases when desired. Graph databases are flexible, and they can be extended to other use cases when needed. Next step towards exploring the possible opportunities, would be to begin working on a selected use case.

REFERENCES

Alfresco*.* Configure databases, website. Available (accessed 18.10.2021): https://docs.alfresco.com/content-services/latest/config/databases/

Angles, R. & Gutierrez C. (2008) Survey of graph database models. *ACM Computing Surveys.* Vol.40(1), Article 1.

Armbruster S. (2016) Welcome to the Dark Side: Neo4j Worst Practices (& How to Avoid Them). Neo4j Blog, website. Available (accessed 5.10.2021): https://neo4j.com/blog/dark-side-neo4j-worst-practices/

Barrasa, J., Hodler, A.E. & Webber J. (2021) Knowledge Graphs: Data in Context for Responsive Businesses. O'Reilly Media, Inc.

Basso, M., Hobert, K. & Mann J. (2016) Gartner Magic Quadrant for Enterprise File Synchronization and Sharing*.* Gartner.

Batra R. (2018) *SQL Primer*. Apress.

Bechberger D. (2019) A Skeptics Guide to Graph Databases. NDC Conferences, video. Available (accessed 1.10.2021): https://www.youtube.com/watch?v=yOYodfN84N4

Bernstein, P.A. & Newcomer E. (2009) Principles of Transaction Processing, Second Edition. Morgan Kaufmann publications, USA.

Bisson S. (2021) Making sense of Microsoft's graph database strategy. InfoWorld, website. Available (accessed 20.10.2021): https://www.infoworld.com/article/3231658/making-sense-of-microsofts-graph-database-strategy.html

Bogdan Giuşcă. (2005) Map of Königsberg in Euler's time showing the actual layout of the seven bridges- highlighting the river Pregel and the bridges. Wikipedia, website. Available (accessed 20.9.2021): https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Box. (2021) Box named a Leader by Gartner for the third year in a row, website. Available (accessed 22.10.2021): https://www.box.com/resources/gartner-content-services-platforms-mq

Campos, M., angelgolfer-ms, Johnston, J., Enriquez, F.C., Coulter, D., Hult, S.T., Graham, L., Okafor, S., Achieng, M., tlenig, Woods, J., OfficeGSX, Thake, J., Arenas, Y., Bowen, K., Aledor D. Overview of Microsoft Graph. Microsoft, website. Available (accessed 21.10.2021): https://docs.microsoft.com/en-us/graph/overview

Chaki S. (2015) Enterprise Information Management in Practice: Managing Data and Leveraging Profits in Today's Complex Business Environment. Apress.

DB-Engines. DB-Engines Ranking, website. Available (accessed 15.9.2021): https://db-engines.com/en/ranking

Gartner. Gartner Glossary: Enterprise Information Management (EIM), website. Available (accessed 10.9.2021): https://www.gartner.com/en/information-technology/glossary/enterprise-information-management-eim

Gartner. Hyland Reviews, website. Available (accessed 17.10.2021): https://www.gartner.com/reviews/market/content-services-platforms/vendor/hyland-software/reviews?industry=9864

Gebert, H., Geib, M., Kolbe, L. & Brenner W. (2003) Knowledge-enabled customer relationship management: Integrating customer relationship management and knowledge management concepts. *Journal of Knowledge Management*. Vol.7(5), pp.107-123.

Hanns K.-K. (2016) Content Management for the Digital Era: Rethinking Strategies for 2017 and Beyond. Presentation on 30.11.2016 at AIIM ELC London.

Hullavarad, S., O'Hare, R. & Roy A.K. (2015) Enterprise Content Management solutions—Roadmap strategy and implementation challenges. *International Journal of Information Management*. Vol.35(2), pp.260-265.

Hyland. (2021) Hyland Content Services Platform, website. Available (accessed 19.10.2021): https://www.hyland.com/en/platform

Hyland. OnBase, website. Available (accessed 19.10.2021): https://www.onbase.com/en

Hyland. Perceptive Content, website. Available (accessed 20.10.2021): https://www.hyland.com/en/platform/product-suite/perceptive-content

Iqbal N. (2021) Solr: Improving performance for Batch Indexing. Box, website. Available (accessed 22.10.2021): https://blog.box.com/solr-improving-performance-batch-indexing

ISO/IEC 9075-15:2019-Information technology database languages — SQL — Part 15: Multi-dimensional arrays (SQL/MDA). (2019) ISO/IEC JTC 1/SC 32

Kashef, M., Liu, Y., Montgomery, K. & Candell R. (2021). Wireless cyber-physical system performance evaluation through a graph database approach. *Journal of Computing and Information Science in Engineering*, Vol.21(2).

Keymark. Automating Your Business Processes With OnBase Software, website. Available (accessed 19.10.2021): https://www.keymarkinc.com/onbase-software/

Keymark. Basic OnBase Database Maintenance for SQL Server, website. Available (accessed 19.10.2021): https://www.keymarkinc.com/basic-onbase-database-maintenance-sql-server/

Khan, W., Ahmad, W., Luo, B. & Ahmed E. (2019). SQL Database with physical database tuning technique and NoSQL graph database comparisons. *IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC).* IEEE. pp.110–116.

Kumar, R. & Pattnaik P.K. (2018) Graph Theory. Laxmi Publications Pvt Ltd, Bengaluru.
Lei, M., Chu, L., Wang, Z., Pei, J., He, C., Zhang, X. & Fang B. (2020) Mining top-k sequential patterns in transaction database graphs. *World Wide Web*. Vol.23, pp.103–130.

Magomedov, S., Pavelyev, S., Ivanova, I., Dobrotvorsky, A., Khrestina, M. & Yusubaliev T. (2018) Anomaly detection with machine learning and graph databases in fraud management. *International Journal of Advanced Computer Science and Applications*. Vol.9(11), pp.33-38.

Manualzz. Perceptive Content Database Installation and Setup Guide, website. Available (accessed 20.10.2021): https://manualzz.com/doc/28738091/perceptive-content-database-installation-and-setup-guide

M-Files. Key Capabilities, website. Available (accessed 26.10.2021): https://www.m-files.com/key-capabilities/

M-Files. M-Files Terminology, website. Available (accessed 26.10.2021): https://www.m-files.com/user-guide/2018/eng/M-Files_terminology.html

M-Files. Named Access Control Lists, website. Available (accessed 26.10.2021): https://www.m-files.com/user-guide/2018/eng/Named_access_control_lists.html

M-Files. Pseudo-users, website. Available (accessed 26.10.2021): https://www.m-files.com/user-guide/2018/eng/Pseudo_users.html

M-Files. Quick and easy integration of business data and documents, website. Available (accessed 26.10.2021): https://www.m-files.com/key-capabilities/technology-integrations/

M-Files. What is Intelligent Information Management? website. Available (accessed 26.10.2021): https://www.m-files.com/products/intelligent-information-management/

Microsoft. Growing organizational intelligence with knowledge and content in Microsoft 365, website. Available (accessed 21.10.2021): https://resources.techcommunity.microsoft.com/wp-content/uploads/2019/09/Microsoft-Content-Services-WhitePaper-5.pdf

Microsoft. Microsoft Viva and Content Services: Partners, website. Available (accessed 21.10.2021): https://resources.techcommunity.microsoft.com/resources/partners/content-services/

Microsoft. Power BI, website. Available (accessed 13.9.2021): https://powerbi.microsoft.com/en-us/

Neo4j Inc. Comparing SQL with Cypher, website. Available (accessed 3.10.2021): https://neo4j.com/developer/cypher/guide-sql-to-cypher/

Neo4j Inc. Cypher Query Language, website. Available (accessed 28.9.2021): https://neo4j.com/developer/cypher/

Neo4j Inc. Execution plans, website. Available (accessed 26.10.2021): https://neo4j.com/docs/cypher-manual/current/execution-plans/

Neo4j Inc. The World's Leading Organizations Rely on Neo4j, website. Available (accessed 20.9.2021): https://neo4j.com/who-uses-neo4j/

Niu, N., Xu, L.D. & Bi Z. (2013) Enterprise Information Systems Architecture—Analysis and Evaluation*. IEEE Transactions on Industrial Informatics.* Vol.9(4), pp.2147-2154.

Nuxeo. (2021) Designing a Modern Platform Architecture for Content Services: A clear path for moving away from legacy information management systems, website. Available (accessed 14.10.2021): https://www.nuxeo.com/assets/documents/platform-architecture-content-services.pdf

Nuxeo. Database Configuration, website. Available (accessed 17.10.2021): https://doc.nuxeo.com/nxdoc/database-configuration/

OpenText. Best Practices for Using Open Text Integration Center - A Technical Whitepaper, website. Available (accessed 25.10.2021): https://www.opentext.com/file_source/OpenText/en_US/PDF/Best%20Practices%20for%20Using%20Open%20Text%20Integration%20Center%20Whitepaper.pdf

OpenText. OpenText Gupta SQLBase, website. Available (accessed 24.10.2021): https://www.opentext.com/products-and-solutions/products/specialty-technologies/opentext-gupta-development-tools-databases/opentext-gupta-sqlbase

OpenText. Server and Database Upgrade and Migration Services, website. Available (accessed 24.10.2021): https://www.opentext.com/file_source/OpenText/en_US/PDF/opentext-so-server-database-upgrade-en.pdf

Oracle. Graph Database and Graph Analytics, website. Available (accessed 18.10.2021): https://www.oracle.com/database/graph/

Oracle. What is a Relational Database (RDBMS)? website. Available (accessed 19.9.2021): https://www.oracle.com/database/what-is-a-relational-database/

Peart, M. & Coulter D. (2021) What is Microsoft Dataverse? Microsoft, website. Available (accessed 21.10.2021): https://docs.microsoft.com/en-us/powerapps/maker/data-platform/data-platform-intro

Perryman J. & Bechberger, D. (2020) Graph Databases in Action: Examples in Gremlin. Manning Publications Co. LLC, New York.

Pokorný J. (2015) Graph Databases: Their Power and Limitations, In: Saeed, K. & Homenda W. (eds). (2015) *Computer Information Systems and Industrial Management*. *CISIM Lecture Notes in Computer Science*. Vol.9339.

Prusti, D., Das, D. & Rath S.K. (2021) Credit Card Fraud Detection Technique by Applying Graph Database Model. *Arab J Sci Eng*. Vol.46**,** pp.1–20.

Robinson, I., Webber, J. & Eifrem E. (2015) Graph databases: new opportunities for connected data, Second edition. O'Reilly, USA, Sebastopol, California.

Shegda, K.M., Hobert, K., Woodbridge, M. & Basso M. (2016) Reinventing ECM: Introducing Content Services Platforms and Applications. Gartner Research, Gartner.

Software Testing Help. Difference Between SQL Vs MySQL Vs SQL Server (With Examples), website. Available (accessed 5.10.2021): https://www.softwaretestinghelp.com/sql-vs-mysql-vs-sql-server/

Songqing, M., Xiaowei, H., Chengshu, X. & Antonio M. (2020) GREG—studying transcriptional regulation using integrative graph databases. *Database*. Vol.2020.

Szendi-Varga J. (2019) Graph Technology Landscape 2019. GraphAware, website. Available (accessed 24.9.2021): https://graphaware.com/graphaware/2019/02/01/graph-technology-landscape.html

Thakur D. What is DBTG? Architecture of DBTG Model. Computer Notes, website. Available (accessed 18.9.2021): https://ecomputernotes.com/database-system/adv-database/architecture-of-dbtg-model

The Apache Software Foundation. (2006) Apache Lucene - Index File Formats, website. Available (accessed 5.10.2021): http://lucene.apache.org/core/3_6_2/fileformats.html

The Apache Software Foundation. (2022) Solr Features, website. Available (accessed 1.4.2022): https://solr.apache.org/features.html

The Apache Software Foundation. Apache HBase, website. Available (accessed 22.10.2021): https://hbase.apache.org/

The Apache Software Foundation. Apache TinkerPop, website. Available (accessed 1.4.2022): https://tinkerpop.apache.org/gremlin.html

TIOBE Index, IBM, Microfocus, Celent & Accenture. (2019) International Cobol Survey Report. (cited by Hartman T. COBOL blues. Reuters Graphics, website. Available (accessed 15.9.2021): https://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/)

Tyrväinen, P., Päivärinta, T., Salminen, A. & Iivari J. (2006) Characterizing the evolving research on enterprise content management. *European Journal of Information Systems*. Vol.15(6), pp.627-634.

Valdes R. (2021) The Competitive Dynamics of the Consumer Web: Five Graphs Deliver a Sustainable Advantage. Gartner Research, Gartner.

Vicknair C., Macias, M., Zhao, Z., Nan, X., Chen, Y. & Wilkins D. (2010) A comparison of a graph database and a relational database: a data provenance perspective. *Proceedings of the 48th Annual Southeast Regional Conference.* ACM. pp.1–6.

Vitt, E., Luckevich, M. & Misner S. (2008) Business Intelligence. 1st edition. Microsoft Press.

Vossen G. (2009) *ACID Properties*. In: Liu, L., Özsu, M.T. (eds) (2009). Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_831.

Woodbridge, M., Sillanpaa, M., Severson, L. & Nelms T. (2021) Magic Quadrant for Content Services Platforms. Gartner Research, Gartner.

Xu L.D. (2011) Enterprise Systems: State-of-the-Art and Future Trends. *IEEE Transactions on Industrial Informatics*. Vol.7(4), pp.630-640.

Xu, L. D. (2015) Enterprise integration and information architecture: a systems perspective on industrial information integration. 1st edition. [Online]. Boca Raton: Auerbach Publications.

Yuhann N. (2020) The Forrester Wave: Graph Data Platforms. Forrester.