A COMPARISON BETWEEN RELATIONAL DATABASE MODELS AND NOSQL TRENDS ON BIG DATA DESIGN CHALLENGES USING A SOCIAL SHOPPING APPLICATION

SERHAT UZUNBAYIR

JUNE 2015

A COMPARISON BETWEEN RELATIONAL DATABASE MODELS AND NOSQL TRENDS ON BIG DATA DESIGN CHALLENGES USING A SOCIAL SHOPPING APPLICATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF

NATURAL AND APPLIED SCIENCES OF

IZMIR UNIVERSITY OF ECONOMICS

BY

SERHAT UZUNBAYIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

JUNE 2015

M.S. THESIS EXAMINATION RESULT FORM

Approval of the Graduate School of Natural and Applied Sciences

[illegible]

Prof. Dr. Cüneyt Güzeliş
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Turhan Tunalı
Head of Department

We have read the thesis entitled **"A Comparison Between Relational Database Models and NoSQL Trends On Big Data Design Challenges Using A Social Shopping Application"** completed by **Serhat Uzunbayır** under supervision of **Prof. Dr. Brahim Hnich** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Brahim Hnich
Supervisor

Examining Committee Members	**Date:** 29.06.2015
Prof. Dr. Brahim Hnich Dept. of Computer Engineering, İUE	[illegible]
Asst. Prof. Dr. Zeynep Nihan Berberler Dept. of Computer Science, DEU	Zeynep
Asst. Prof. Dr. Senem Kumova Metin Dept. of Software Engineering, İUE	[illegible]

ABSTRACT

A COMPARISON BETWEEN RELATIONAL DATABASE MODELS AND NOSQL TRENDS ON BIG DATA DESIGN CHALLENGES USING A SOCIAL SHOPPING APPLICATION

SERHAT UZUNBAYIR
M.S. in Intelligent Engineering Systems
Graduate School of Natural and Applied Sciences
Supervisor: Prof. Dr. Brahim Hnich
June 2015

Data generation is increasing excessively day by day. Consequently, the term Big Data came out to expand the meaning of data we face nowadays. Traditional database technologies started to have struggles when operating the applications containing huge amount of data. Such problems impelled researches to develop brand new ways of handling data.

All systems should try to adopt changes resulting from new requirements when necessary. There are various database management systems and products in the market. Relational databases were efficient to store and process data since 1970s. However, today's amount of data is far more huge compared to even last couple of years ago. This situation inevitably forces some systems to shift their design from relational models to NoSQL trends. There are also various NoSQL technologies exist with a number of different products developed by companies. In this case, developers may be confused to decide which type of database should be used in order to deal with Big Data and its problems within their systems.

In this thesis, we summarize database management systems including NoSQL and challenges among them. We analyse and compare two different database technologies in detail; relational and graph databases. We design and develop data models for both technologies for a social shopping system called TrendPin. We show design models as well as distinct query performances. Additionally, we explain information extraction process and implement a knowledge base for TrendPin to overcome problems we encountered when designing graph model.

Keywords: database, big data, sql, nosql, database management systems, relational database, graph database, information extraction, knowledge base, feature extraction, online shopping.

ÖZ

BÜYÜK VERİ TASARIM ZORLUKLARI ÜZERİNE BİR SOSYAL ALIŞVERİŞ UYGULAMASI KULLANILARAK İLİŞKİSEL VERİTABANLARI VE NOSQL AKIMLARI ARASINDA BİR KARŞILAŞTIRMA

SERHAT UZUNBAYIR
Akıllı Mühendislik Sistemleri, Yüksek Lisans
Fen Bilimleri Enstitüsü
Tez Danışmanı: Prof. Dr. Brahim Hnich
Haziran 2015

Veri yaratımı günden güne fazlaca artmaktadır. Sonuç olarak bugünlerde karşı karşıya kaldığımız verinin anlamını genişletmek için Büyük Veri terimi ortaya çıktı. Geleneksel veritabanı teknolojileri büyük miktarlarda veriler içeren uygulamaları çalıştırırken mücadeleler yaşamaya başladı. Bu tür sorunlar araştırmacıları yepyeni veri işleme metotları geliştirmeye yöneltti.

Tüm sistemler gerekli olduğu zaman yeni gereksinimlerden kaynaklanan değişimlere ayak uydurmak zorundadırlar. Piyasada çeşitli veritabanı yönetim sistemleri ve ürünleri bulunmaktadır. İlişkisel veritabanları 1970'lerden beri veri saklama ve işleme konusunda etkiliydiler. Fakat bugünkü verinin miktarı geçtiğimiz birkaç yıla göre karşılaştırıldığında bile çok fazladır. Bu durum kaçınılmaz olarak bazı sistemlerin tasarımlarını ilişkisel modellerden NoSQL akımına çevirmeye zorlamıştır. Farklı şirketler tarafından geliştirilen bir çok farklı NoSQL ürünü vardır. Bu durumda geliştiriciler sistemlerindeki Büyük Veri ve onun problemleri ile uğraşmak için hangi tür veritabanı seçeceklerine karar vermekte zorlanmaktadırlar.

Bu tezde, farklı veritabanı yönetim sistemleri ve zorlukları özetlenmiştir. İlişkisel ve grafik tabanlı olmak üzere iki farklı veritabanı teknolojisi analiz edilmiş ve karşılaştırılmıştır. Bu iki teknoloji için sosyal ağ ile çevrimiçi alışveriş uygulaması olan TrendPin üzerinde veri modelleri tasarlanmış ve geliştirilmiştir. Tasarım modelleri ve farklı sorgu performansları gösterilmiştir. Ek olarak bilgi çıkarımı konusu açıklanmış ve grafik modeli tasarımında karşılaşılan sorunların üstesinden gelmek için bir bilgi bankası oluşturulmuştur.

Anahtar Kelimeler: veritabanı, büyük veri, sql, nosql, veritabanı yönetim sistemleri, ilişkisel veritabanı, grafik veritabanı, bilgi çıkarımı, bilgi bankası, özellik çıkarmı, çevrimiçi alışveriş.

ACKNOWLEDGEMENT

I would like to take the opportunity here to thank some special people, who motivated me and gave me the strength to close one chapter of my life successfully.

First of all, I would like to thank to my advisor, Prof. Dr. Brahim Hnich, who gave me the opportunity and privilege to work with him. He is one of the most brilliant person I have ever met in my life. His ability to point out the right way all the time, endless support, supervision, and patience guided me until the end of the road we started together. He was always there when I needed him, even from abroad.

Secondly, I must express my thanks to Ata Şaşmaz for each and every second of his efforts to help me, and Özkan Sayın for his unlimited advices, and encouragements when I was confused.

I would like to thank very specially to my colleagues Erdem Okur, Levent Tolga Eren, and Berkehan Akçay for their trust, encouragements, and never ending supports during the process of this thesis.

I am very thankful to all of my friends. This world is worth living because of them.

Finally, I am very thankful to my father Ömer, my mother Gül, and my beloved sister Gülşah for their love and unconditional support all the time.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Topic of the Thesis

The era we live in today allows technological systems to keep their importance at high levels. With the help of these systems, data on the paper is being digitalized to facilitate easier and much faster data generation. Digital data is being created with only one single touch on mobile phones, tablets, or computers and it can be used immediately though the Internet. Development of smart devices and especially the introduction of social networks have dramatically affected the increase in the amount of data. Disturbingly, these advancements altered the type of data and revealed the notion of Big Data.

Many applications in different domains from military to social media and even scientific research can create Big Data. Militaristic applications are trying to find and locate threats using large volumes of incoming data from various sources, each second thousands of pictures and videos are being uploaded to social media applications such as Facebook and Instagram from all over the world, cameras are constantly observing traffic or certain places, sensor data is coming from RFID tags, statistical data is being collected for economic sources, scientific research on biomedical and health care areas are being done by scientists. It is obvious that data generation is visibly increased.

Databases are collections of data in an organized way to make storing, analysing, and retrieving possible. Normally traditional database procedures store data in a relational table fashion. When new data is created, it can be added to the related row on the table. There are different relational database management products. The most popular examples are; *MySQL*, *Oracle*, and *Microsoft SQL Server*. In order to fetch data, queries should be written and executed. Those products use Structured Query Language (SQL) for this purpose. SQL allows many data manipulation operations such as insertion, update, and deletion operations.

Relational database management systems are quite good at handling small-sized data. However, as data generation grows huge, storing and analysing operations become quite complicated for relational models. There may not be enough storage to hold new data if database is centralized, or it may not be not efficient to separate useful information among whole data stack. Usual data models are no longer able to stay their positions at the center of the area. This is where not only SQL (NoSQL) approaches show up and help to overcome those kinds of issues regarding to Big Data.

Time is one of the precious dimensions of daily life. Of course, the most time consuming action is working for many of people. Busy business life definitely prevents people to spend time going from one shop to another for specific needs. Even for non-workers or when the shops are closed, it is a brilliant idea to be able to start or continue shopping using online shopping applications. Invention of the Internet made that idea possible and broader with companies such as *amazon*, *e-bay*, and *gittigidiyor*. On the other hand elements of social media and friendship among users create a suitable environment to expose personal information, feelings, opinions, news about their lives as long as they allow other users to follow them. When these two distinct formations are mixed up together, a brand new online shopping with social networking platform TrendPin came out.

There are many different NoSQL approaches; key-value pairs, column-based databases, document databases, and graph databases. This thesis briefly explains different NoSQL types and their products. Since social networking is producing

Big Data, and can easily be modelled with graphs, graph databases are appropriate to handle storing and querying such information over distributed servers. To accomplish this structure, we created a graph model for TrendPin in this thesis work and compared its performances with relational data model.

During graph modelling of TrendPin, there were some problems stemmed from the project specification when converted into graph model. We present feature extraction processes with their importance to increase performance of the systems. We implemented a knowledge base for TrendPin and explain why it was necessary to build include such structure at the end.

1.2 Contributions of the Thesis

In this thesis we discuss what Big Data is, why it carries such importance, and what kind of problems occur during generation, storing, and retrieving. We explain various types of NoSQL approaches and choose graph databases to compare with relational database models. We analyse relational model of Microsoft SQL Server, graph model of Neo4j, and create two versions of TrendPin to support these products. We discuss strengths and weaknesses of both models for same requirements and point out performance issues.

Relational models differ from graph models in both design and data processing stages. During graph modelling we encountered problems about connections between same products but from different vendors, and connections between related products. Also we wanted to display much smarter results to the users when searching a product. To solve those problems, we investigate information extraction processes and present a knowledge base for the project, explain benefits, and assess increase in performance of the system.

Main contributions of this thesis are as follows;

- We discuss types of NoSQL briefly, explain graph databases in detail, discuss different products on the market for each type.

- We present two different data models, relational model and graph model, for TrendPin, and show how to create those models for given requirements.
- We explain information extraction processes, why we need to use these techniques for this project, and present a knowledge-based information retrieval module for TrendPin to overcome some design problems.
- We present different queries based on same purposes for both versions of the system, and compare performance results.
- We provide a comparison on design challenges such as maturity and resilience for both models.

1.3 Outline of the Thesis

The outline of this thesis is as follows;

- Chapter 1 gives an introduction and overview of the thesis as well as thesis plan.
- Chapter 2 presents the definition of Big Data, its characteristics and challenges in the field. In addition, it discusses how to move from relational models to distributed approaches.
- Chapter 3 discusses traditional database technologies and explains NoSQL trends by giving examples and acquaints different products.
- Chapter 4 introduces TrendPin; a social shopping platform.
- Chapter 5 discusses various database approaches implemented on TrendPin by showing data models. Additionally it explains how the implementation was being done.
- Chapter 6 explains information extraction methods and presents a knowledge-based information retrieval module for TrendPin in order to fix design issues originated from its graph model.

- Chapter 7 discusses experimental results over queries in terms of performances, and discusses design challenges of both models in terms of specific properties.
- Chapter 8 summarizes and concludes the thesis by stating future work.

Chapter 2

Big Data

2.1 Big Data Definition

Today perhaps the most important value of mankind is information. People can move forward during their life with information. As for the computer science, collection of information is said to be data. Invention of the internet and social media, with the help of mobile devices, every day a lot of data is being generated in each and every domain of life varying from application domains such as mobile applications to improve healthcare, to educating online without ever going to any other institutions.

Al-Khouri (2014) [34] explains on Figure 2.1 that there were 160 exabytes of data in all hard drives in 2006. After 2007, growth of data increased exponentially. And expecting results proposes that this growing is going to reach 112 zettabytes of data by 2020.

Yan (2013) [87] states that Big Data term is emerged during 1980's academic and industrial meetings. She also adds that there were dozens of definitions discussed throughout the literature. However, Diebold (2000) [53] covered academic definition of Big Data for the first time by stating "Big Data refers to the explosion in the quantity (and sometimes, quality) of available and potentially relevant

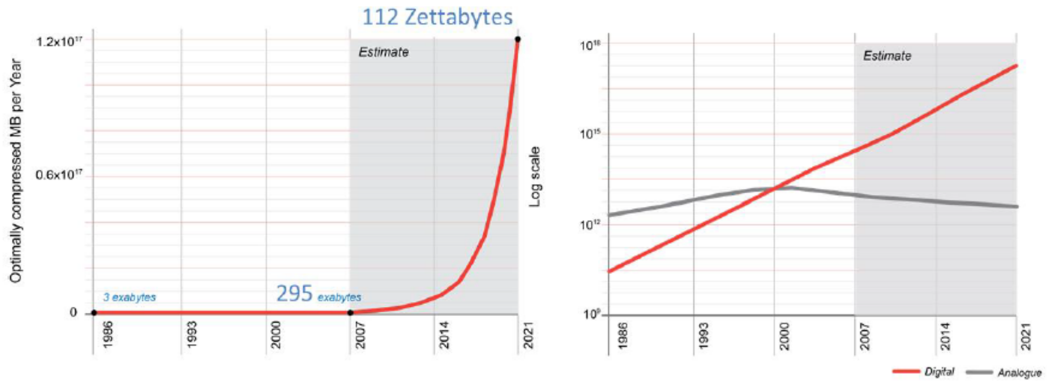


Figure 2.1: Growth of Data Over the Years [34]

data, largely the result of recent and unprecedented advancements in data recording and storage technology." Not only quantity, but also the rate of accumulation of data is reached to the levels which cannot be ignored.

Maier (2013) [74] explains that many organizations extended the definition of the term. For instance Smith (2012) [81] told that "'Big Data' represents the historical debri (observed data) resulting from the interaction of at between 70 and 77 independent variable/subjects". He also adds that these subjects can be both instances of unknown populations which are not randomly generated, and moving along aimed time slots. As data becomes unstructured with the increase in its type, the definition becomes much clear.

Big Data is also characterised by a change in the qualities of data held by organisations. In traditional methods, data is stored in a highly structured format to maximise its informational content. For example, a relational database has a set number of fields, each of which will contain a specific type of data in a specific format. This structure makes it easy to process and manipulate by applying simple deterministic rules to that data. However, Big Data is not easy to store and handle efficiently within relational tables because of its characteristics. With the demonstration of the meaning made Big Data became quite popular and researchers started to investigate the topic quickly.

2.2 Characteristics of Big Data

The data that cannot be managed by relational database management tools is not enough to explain and understand what Big Data is. To be able to give the sense of its true definition, the characteristics should be discussed. Doug Laney (2001) [70] introduced 3V's (Figure 2.2) concept for the first time to characterize Big Data; volume, variety, and velocity.

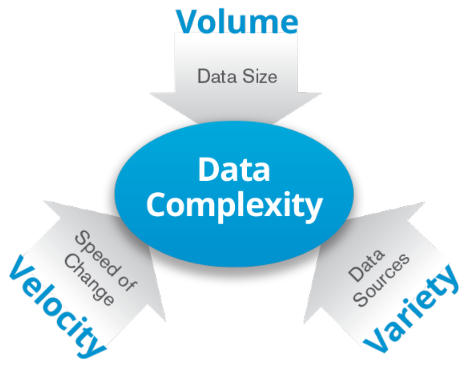


Figure 2.2: 3V's of Big Data [48]

2.2.1 Volume

The word *"big"* in the term Big Data sounds like the data is not small at first glimpse. However, *"how big is this Big Data?"* is a key question. Huddar and Ramannavar (2013) [67] claim that 800,000 petabytes of data were stored in the world in 2000. And they state that in 2020, this number is going to reach 35 zettabytes. Additionally, with the growth of social media, and the products of the companies which are selling data management services increase this number everyday. A recent report from IBM (2011) [33] declares that 90 % of the data has been generated over the last two years. Too much data has been generated and is continuing to be generated each second, yet most of it still not analysed at all.

An intense research of a company named Domo [9] puts forward how much data is being generated in every minute from users of different companies. The research claims that;

- YouTube users upload 48 hours of new video,
- Email users send 204,166,667 messages,
- Google receives over 2,000,000 search queries,
- Facebook users share 684,478 pieces of content,
- Twitter users send over 100,000 tweets,
- Instagram users share 3,600 new photos,

These facts uncover that many organizations are generating zettabytes of data every day. Many companies are having problems with storing huge amount of data because they cannot store using traditional database management systems. They want to analyse and extract the useful data, gain benefits from it to understand how well the business is going on, or customer thoughts and etc. When the amount of data is growing, the ability of processing of data is decreasing. To overcome this issue, they need to use the right technology.

2.2.2 Variety

As the volume of data increases, the variety of data changes. The usage of especially mobile devices, social media, as well as different types of sensors is increasing every other day. “As new services are added, new sensors deployed, or new marketing campaigns executed, new data types are needed to capture the resultant information” (Dijcks, 2013) [54]. These actions allow users to produce many different kinds of data; social data, statistical data, medical data, surveillance data. It can be understood that this data is not always in text or numerical format. Huddar and Ramannavar (2013) [67] states that it can be raw data, semi-structured data, or unstructured data from web pages that contain image, video, audio, pdf, web log files, sensor readings and etc.

Variety of data also means the variety of databases. Traditional database management systems are having difficulties when it comes to analyse different kinds of data in a single query. Storing this data is also creates difficulties for traditional methods. For example video and image files cannot be efficiently stored in a relational database. When the value of an entity is changing frequently, it is not very efficient to apply changes on a scheme. Since traditional database management systems are not efficient when it comes to process various kinds of data, many organizations are still having struggles about managing, merging and governing different varieties of data.

2.2.3 Velocity

Normally data is not being generated in an ordered format. With the broad usage of internet in mobile device services increased data generation rate exponentially. Data is now generated at any time and at anywhere from mobile devices, sensor networks, and such tools. Here the characteristic term velocity is about how quickly data is arriving and stored, and its associated retrieval. It is important to get the data as fast as possible, manage it, and convert it to output for taking the best and the most efficient feedback. Many companies are keeping record of every single transaction especially on their websites. Dumbill (2012) [56] claims that, online retailers are able to compile large histories of customers every click and interaction: not just the final sales. They are doing this so that they can recommend similar products by quickly suck advantage out of customer information and go one step ahead in the market. Another important point is by using sensors and smart devices, data streaming happens too fast and in near-real time. The new designed systems have to react to the changes as quick as possible to be able to answer the needs of the organizations.

2.2.4 Fourth Addition to 3V's: Veracity

Some researchers add another characteristic onto 3V's called "veracity" which is explained as uncertain, noisy, or imprecise data. There is not a single reason to have uncertain data. For example there maybe same attribute name that corresponds to different entities, or vice versa (Maier, 2013) [74] Figure 2.3 explains three dimensions of veracity.

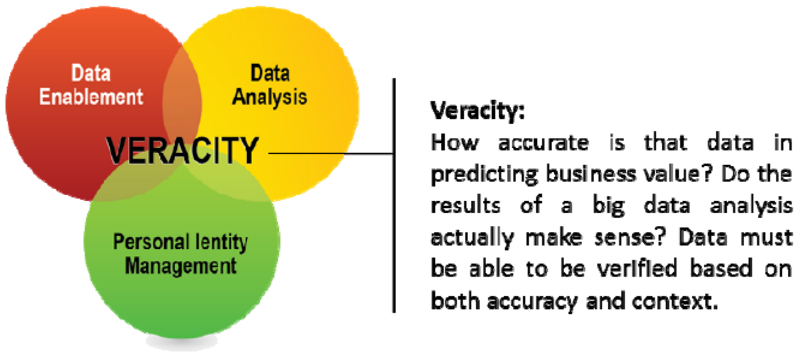

Data Enablement
Data Analysis
VERACITY
Personal lentity Management
Veracity:
How accurate is that data in predicting business value? Do the results of a big data analysis actually make sense? Data must be able to be verified based on both accuracy and context.

Figure 2.3: The three dimensions of veracity (Al-Khouri, 2013)[34]

2.2.5 3C's Concept

According to Suthaharan (2013) [83], when a set of data is growing to infinity by having k number of zeros, ones, twos, threes, and etc. will be referred as Big Data within 3V's space. However, when a sample is taken from such data will still be a small data. By claiming this argument, he suggests new characteristics of Big Data and calls them as C3; cardinality, continuity, and complexity. He then defines these three new properties;

- Amount of records in the dataset which grows dynamically as *cardinality*,
- Increase of data size in time with a continuous function as *continuity*,
- Variety of data, huge volumes of dataset, and the high speed of data processing as *complexity*.

2.3 Sources of Big Data

Big data is nearly everywhere and many systems are generating it without even being stopped once. There are plenty sources in several domains that increases data traffic through the cloud continuously. Now we will discuss some of these sources.

2.3.1 Social Networking

Perhaps the most popular data source of Big Data is social networking. Today the Internet can be accessed from almost everywhere through a number of devices. This allows people to connect with others when they are on the road, or at home, or while doing their jobs. They can chat, request support for an issue, send/check job applications, check locations, play games together, and etc. According to a study of Pew Research Center in Figure 2.4, more than 70% of all internet users use social networking sites. Same study also indicates that there is an increase on not only the numbers of adult users, but also the number of middle-aged and elderly users who use social media. There are different applications, such as Instagram, Twitter, Facebook, Snapchat, Vine, Youtube, Super, to connect with other users. Type of the generated data can be various too. Some applications just create images, some of them create videos, some of them create texts, and some of them create mixture of all.

Social media is sure plays an important role when connecting with people. On the other hand, it enables;

- Free advertising in front of billions of people,
- Marketing of companies and products,
- Product promotions,
- Predictions of events like the results of football matches, political situations, new music album successes,

- E-learning,
- Fast spread of news, and many more.

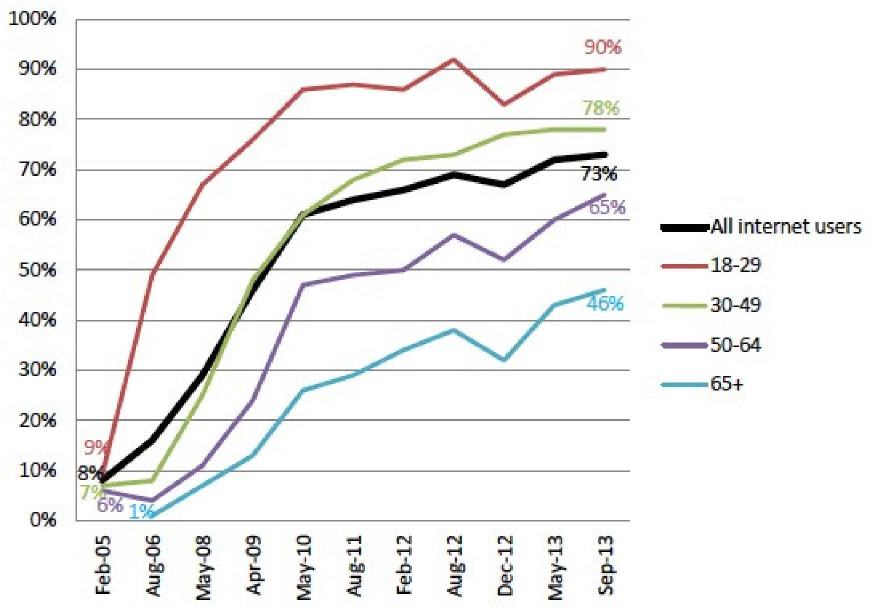


Figure 2.4: Social networking site usage by age group, 2005-2013[64]

2.3.2 Sensor Data

A sensor is a hardware whose task is to sense its environmental phenomenons e.g. heat, sound, motion, and transfer its findings. Sensors are distributed to every corner of the world and gathering real-time data for certain objectives. Some of the examples of such objectives are;

- Monitoring health of patients with wearable wireless sensor devices to find symptoms of clinical diseases, or tracking medical conditions.
- Observing traffic on the roads with image sensors in order to report current traffic situation, or even find and locate criminals.
- Measuring weather conditions and predict daily and weekly forecasts.

- Monitoring environments for specific purposes such as to measure heat increase around active volcanoes and start evocations of residential areas.
- Observing animals for wildlife tracking.

2.3.3 Online Transactions

Transactional data is the data which is generated by a transactional event. The difference between non-transactional data and transactional data is time dimension. Transactional data always has a time variable to indicate occurrence of the event. For example; an order from a website, billing, online trading, and internet banking processes are create transactional data. According to a research [20], online retails are increasing (see Figure 2.5). This proves that transactional data is blowing all over the internet and will continue to increase in the near future.

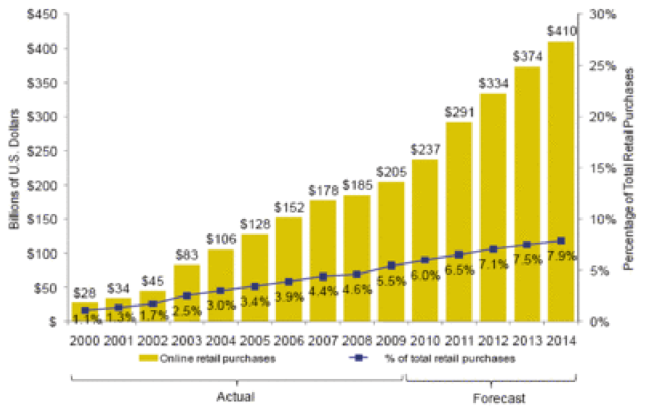


Figure 2.5: E-business transaction volume and online retail purchases ratio

2.3.4 Smart Devices

Changes in the technology direct to turn more devices into smart devices. Smart phones, smart watches, smart televisions, smart glasses are very popular and becoming indispensable items of daily life. There are many functions of those technologies. By using smart phones, we can do a lot of stuff from paying bills to find places. Smart watches can now be used to listen to radio and call others. Smart televisions provide surfing on the internet, watch online films. Smart glasses allow recording videos with just a blink of an eye. All of are just examples and it is thrilling to predict what comes next in the future.

2.4 Challenges of Big Data Services

Big Data creates new opportunities to the market, and opens a whole new research area to the researchers. At the same time the services held by Big Data based systems have many computational challenges. Laney (2001) [70] states that the challenges are mostly arise from complexity that is caused by three dimensional characteristics; volume, variety, and velocity.

Since the volume of the data is very big, it needs to be decided how to store it properly. When it comes to variety, it has to be analysed while keeping in mind that unstructured data can contain various sources in various formats such as audio or video files. Velocity is related to generation speed of data. It should be generated in real time and have to be managed within certain time intervals before delivery.

According to Fan et. al. (2013), characteristics of Big Data allows many different challenges ranging from unique statistical and computational issues. These challenges include "scalability and storage bottleneck, noise accumulation, spurious correlation, incidental endogeneity, and measurement errors"[58].

Apart from those challenges above, Cloud Security Alliance (2012) [35] reports that there are also security and privacy challenges such as data provenance, endpoint input validation, real-time security monitoring and etc.

Big Data challenges sometimes differ from domain to domain. Dobre and Xhafa (2013) [55] suggest that in a context-aware platform for large scale data handling, there are some problematic requirements that should be handled by Big Data services such as mobility and locality, proximity, real-time guarantees, support for communication imperfection, and etc.

Because of the novelty of the concept and research area, there are other challenges such as lacking efficient tools and experienced big data developers.

2.5 Big Data Storage and Access

Data is expanding in three dimensions day by day. This growing inevitably pushed database management system developers to change the way that they work with the latest researches and developments. Angeles and Castro (2013) [37] states that most database management systems are either row based (horizontally scalable) or column based (vertically scalable) shown in Table 2.1. However, as Angadi et. al. (2013) [36] discussed that the organizations, with the new projects under development, are going away from *"one size fits all"* approach because of the challenges that Big Data has brought to the field.

ID	**Name**	**Age**
1212	Jack	25
1414	Caroline	27
3214	Klaus	18

Row 1	1212
	Jack
	25
Row 2	1414
	Caroline
	27
Row 3	3214
	Klaus
	18

ID	1212
	1414
	3214
Name	Jack
	Caroline
	Klaus
Age	25
	27
	18

Table 2.1: Table (left), Row (center), Column (right) based database structures

Storing Big Data with tables on a single machine is not efficient, due to the fact that data growing speed is tremendously fast. On the other hand, it is not always very easy to develop a great structure for distributed machines working over cloud and expect it to perform as promised. Timely design considerations should be taken care of very well. There are three important properties to be considered when developing such systems:

- **Performance** is the first property. When an input is given to the system, the results should be received as fast as possible. The algorithms are the most effective factors on this issue. Problems can be different, and they should be coded specifically to the problem. If it is a centralized system, performance will be relying on the hardware. However, when the system is distributed, a huge effector called *latency* will come to the playground. Latency is the time taken between a response and its stimuli. Mostly, latency should be very low, in case of improved performance. However, network connections are not always stable. This causes problems on real-time applications especially. For example, when a user is taking an online exam using a computer, questions may not be answered in time, if latency is high. This may cause him to fail not because of his knowledge, but because of the system's performance. Probably the main reason of this issue would be derived from communication overheads, or the server may be very far away from the user's location. During design phase, it should not be ignored in that those overheads should be reduced.

- **Scalability** is another property. It means that a system should continue operating when the size of inputs are very large and coming fast. To do so, it is preferred to increase storage automatically. According to the type of the system design, scalability can be divided into two:

  - *Vertical scaling* means that the system's storage is increased with the addition of more memory or CPU of the computer on centralized systems.
  - *Horizontal scaling* means that the system's storage is increased with the addition of new nodes working over cloud on distributed systems. Adding new nodes expands the storage of the system automatically and there will not be any information loss. In our case, this type of scaling is preferred, and it increases the performance of overall system.

- **Availability** is the last property. Systems should be available at any time when a request has been created. For example, sometimes ATM machines may not be available; credit card or money cannot be inserted because of hardware faults. For distributed systems, in order to prevent information losses, data should be copied and stored on different servers. With this way, even though a server is not available, the operation can be done using another server which contains same information.

It is obvious that distributed systems have more advantage over centralized systems in terms of performance, scalability, and availability when storage is the concern. Since Big Data requires distributed approaches to perform efficiently, the main system which inspired others on the market is called Apache's Hadoop Distributed File System (HDFS). Before explaining it, we are going to define a data processing paradigm called MapReduce, which became popular after 2004 and inspired HDFS.

2.5.1 MapReduce

In early 2000s, Google and some other companies were having struggles to keep vast amounts of data in a single database. To overcome this serious issue Google presented MapReduce programming model (Dean and Ghemawat, 2004) [50]. It is a programming paradigm to process large volumes of data within distributed clusters on the cloud using parallel programming style. Simply, the model takes key/value pairs as inputs and processes them to output key/value pairs. The process involves two distinct operations; **Map**, and **Reduce**:

- **Map** operation converts inputs to intermediate forms and groups them.
- **Reduce** operation takes these intermediate key/value pairs and combines them to create smaller forms.

After **Map** operation produces a list of key/value pairs, these values are sent to **Reduce** operation to combine and create outputs. MapReduce process requires five distinct stages to produce associated output: *Splitting*, *Mapping*, *Shuffling*, *Reducing*, and *Finalizing*.

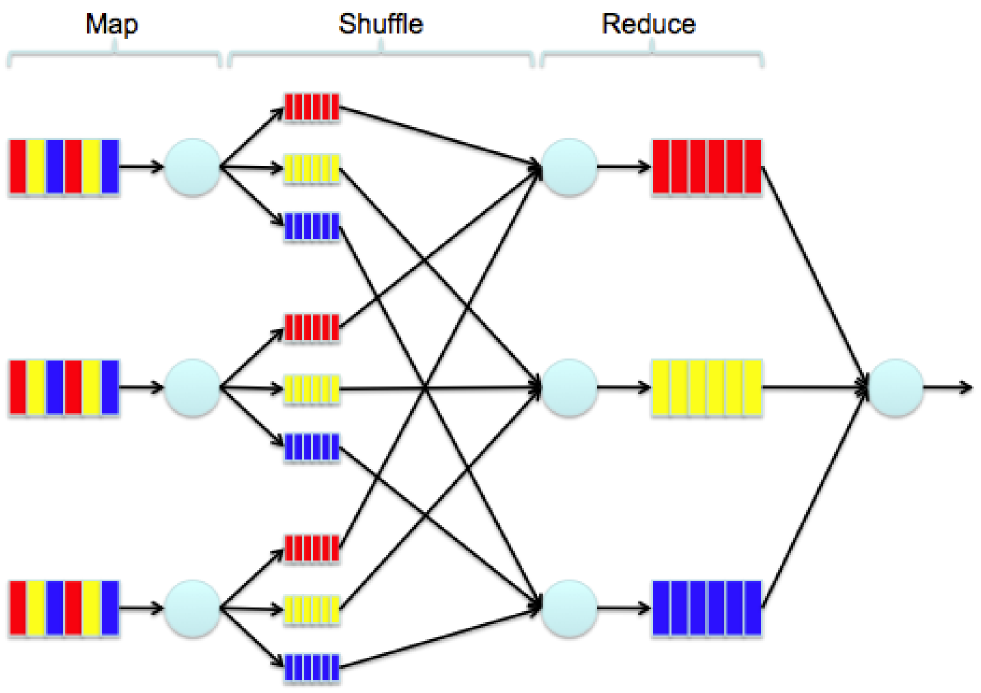


Figure 2.6: Example of a Map Reduce Model [71]

Let us explain how MapReduce process takes place with an example. Assume that we have the following two samples of data taken from a course statistics table:

Course Code	**Number of Passed Students**	**Year**
SE 315	39	2012
SE 305	73	2011
SE 315	44	2011
SE 115	144	2011
SE 305	60	2012
SE 115	148	2012

Course Code	**Number of Passed Students**	**Year**
SE 305	40	2014
SE 315	72	2013
SE 305	52	2013
SE 115	152	2014
SE 115	168	2013
SE 315	81	2014

Table 2.2: Sample data collection about the number of students and courses

Sample data above shows three different courses (SE 115, SE 305, and SE 315), and the number of students who passed these courses from 2011 to 2014 separately. Now we want to find the **total** number of students who passed **each** course between 2011-2014. Lets explain whole MapReduce process step by step:

- **Step 1 - Splitting:** The first operation is to split given datasets into 10 distinct rows.

SE 315	39	2012

SE 305	73	2011

SE 315	44	2011

SE 115	144	2011

SE 305	60	2012

SE 115	148	2012

SE 305	40	2014

SE 315	72	2013

SE 305	52	2013

SE 115	152	2014

SE 115	168	2013

SE 315	81	2014

- **Step 2 - Mapping:** Now *Map* function will generate a list of key/value pairs. In this case we want to have course name as keys, and number of passed students as values.

SE 315	39
SE 305	73
SE 315	44
SE 115	144
SE 305	60
SE 115	148

SE 305	40
SE 315	72
SE 305	52
SE 115	152
SE 115	168
SE 315	81

- **Step 3 - Shuffling:** The next step is to shuffle all pairs to get same keys together before reducing them to single key/values pairs.

SE 115	144
SE 115	148
SE 115	152
SE 115	168

SE 305	73
SE 305	60
SE 305	40
SE 305	52

SE 315	39
SE 315	44
SE 315	72
SE 315	81

- **Step 4 - Reducing:** Now *Reduce* function will perform reducing operation and all courses will be grouped with the associated values as only one row.

SE 115	{144, 148, 152, 168}

SE 305	{73, 60, 40, 52}

SE 315	{39, 44, 72, 81}

- **Step 5 - Final result:** In this last step, the system will aggregate the values within their keys and display as the final output.

SE 115	612

SE 305	225

SE 315	236

Gu et. al. (2014) [60] summarizes the benefits of MapReduce technique; “MapReduce significantly simplifies the design and implementation of many data-intensive applications in the real world. Moreover, MapReduce offers other benefits, including load balancing, elastic scalability, and fault tolerance, which makes it widely adopted parallel computing framework.”

2.5.2 Apache’s Hadoop Distributed File System

MapReduce opened a whole new world for Big Data storage systems and it became an indispensable component of Hadoop. In 2006, Doug Cutting, an employee of Yahoo, intended to create a web search engine called Nutch, and developed Hadoop. It is a framework written in Java that contains Hadoop Distributed File System (HDFS) and MapReduce components, and aimed to manipulate the data on clusters consist of commodity hardware (White, 2010) [85]. “It enables applications to work with thousands of computational independent computers and petabytes of data.” (Kiran et. al., 2013) [69].

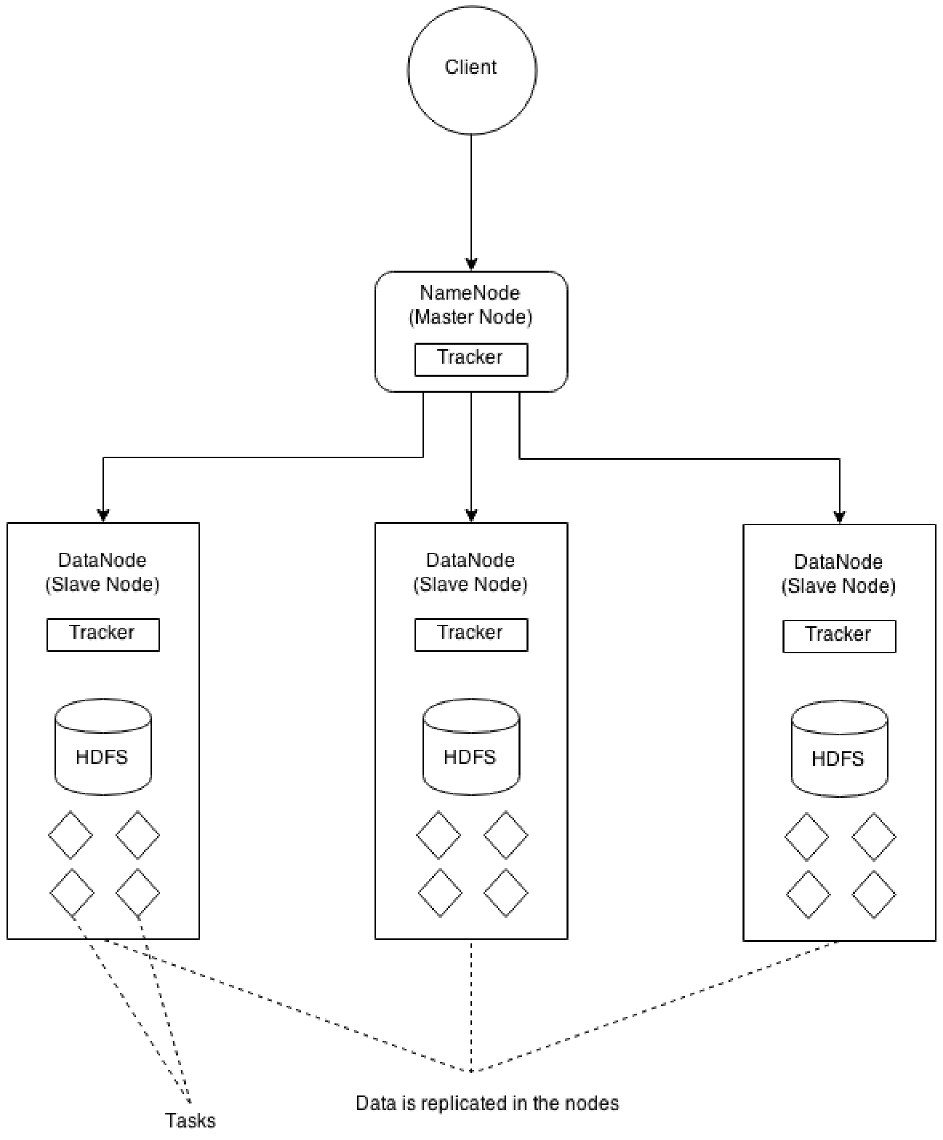


Figure 2.7: Hadoop file structure

HDFS inside Hadoop is able to contain many and very large files by creating clusters of commodity hardware nodes. All files are replicated among different nodes, in order to prevent data losses. HDFS works as a master-slave model and its structure includes **NameNode** as a Master Node and **DataNode** as a Slave Node. There can only be one NameNode and many DataNodes in each cluster.

NameNode is responsible from file creations, deletions, and modifications. It also controls accessibility, transmission of those files when requested from DataNodes as well as creating replicas among nodes. DataNodes are responsible from their local disks.They can contain replicated data of other DataNodes.

MapReduce technique of Hadoop is called Hadoop MapReduce. The method is the same with MapReduce idea we discussed in the previous section. A Map function is required to filter the data, and a Reduce function is required to gather results from filtered data. On the other hand, the role of Hadoop here is to connect DataNodes under a NameNode and perform MapReduce operations to manage Big Data.

Rao et. al. (2013) [79] reminds that Yahoo, Facebook, LinkedIn, and Twitter uses Hadoop for important part of their services. There are other products which use Hadoop to perform their own operations;

- *HBase* [13], a non-relational database model runs on HDFS.
- *Hive* [14], a data warehouse runs on top of Hadoop.
- *Zookeeper* [32], a distributed configuration, synchronization, and naming service.
- *Ambari* [4], a monitoring and managing tool for Hadoop clusters.
- *Pig* [22], a platform to create MapReduce programs.

2.5.3 Summary

Type of the data is changed forever. It is now big, even getting bigger from three dimensions; volume, variety, and velocity. There is nothing to stop Big Data generation. However, it is very important to continue discovering new ways of analysing and handling it. It can be seen in nearly every environment; in social life for communicating, in marketing to reach wider customers for presenting new products, spreading news to anywhere from all over the world, monitoring places, collecting environmental information, and for many more purposes. From all of those unstructured data, each information pieces would be very valuable to different organizations.

Since it is not easy to store and handle Big Data, it is a must to change the way to look at traditional database management systems. They are not enough and efficient, because there are challenges revolving around the concepts of performance, scalability, and availability. Reading data, writing data are not very fast enough so that new approaches are becoming popular. MapReduce techniques from Google enlightened other companies in this sector to develop more efficient products. Apache's Hadoop Distributed File System and the other products stabilized the quality of Big Data storage and processing.

Chapter 3

Relational Databases and NoSQL

3.1 Introduction

Databases can be defined as collections of organized information. This information can vary from numeric entities to image entities. In addition to those, there are relations to create connections between one entity to another. A database is modelled according to some defined purposes. For example; to store information about students in a university, or to keep track of product sales in a company, or even to record user information of a gym center.

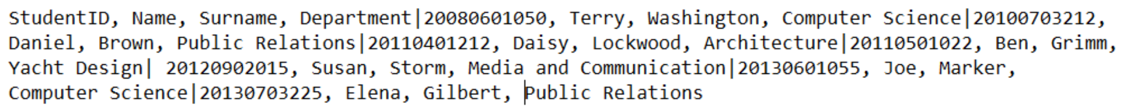
A database requires software applications called database management systems, in order to manipulate stored information. They provide easy reading and writing procedures such as creating, inserting, deleting, and updating. Additionally, by using combinations of these procedures, a database can be backed up and maintained when necessary to restore if any information is lost during database usage.

Database management systems are everywhere and have many advantages. They use powerful methods to hold information together and provide quick ways to retrieve when requested. When Big Data is the main circumstance, traditional methods became no longer that powerful. Large datasets started to not only outgrow existing hard drives, but also slow down data fetching speed. Therefore, new ways of handling data became an important concern of researchers.

Each database management system should be modelled regarding to the requirements that are coming from system's specifications. With respect to data storing, retrieving, and handling methods, database management systems are classified into several types. We are going to explain relational databases and each one of NoSQL trends.

3.2 Relational Database Management Systems

Relational databases are first mentioned in a research paper of a computer scientist Edgar Frank Codd[45], who was working for IBM during 1970. The paper broke a fresh ground for databases and literally redefined the ways of data management. Before that, databases were written in flat files; huge single text files where the data is being separated by particular characters such as commas. Here s an example of a flat file:



```
StudentID, Name, Surname, Department|20080601050, Terry, Washington, Computer Science|20100703212,
Daniel, Brown, Public Relations|20110401212, Daisy, Lockwood, Architecture|20110501022, Ben, Grimm,
Yacht Design| 20120902015, Susan, Storm, Media and Communication|20130601055, Joe, Marker,
Computer Science|20130703225, Elena, Gilbert, Public Relations
```

Figure 3.1: Example of a flat file

In the example above, all information is given as a one single line. It looks complicated and not understandable when read if the size is huge. The file needs to be searched from start to end each time for finding a specific entry and this was not an efficient way to do it.

After relational databases have been introduced, many companies started to use them and they became quite popular since then. They maintained their position in the market for forty years. When Big Data came out, developers realized that relational model cannot adopt itself to work efficiently towards fast growing, and fast scaling data. Multiple join operations, nested queries reduced the performance of such systems.

StudentID	**Name**	**Surname**	**Department**
20080601050	Terry	Washington	Computer Science
20100703212	Daniel	Brown	Public Relations
20110601212	Daisy	Lockwood	Computer Science
20110501022	Ben	Grimm	Yacht Design
20120501015	Susan	Storm	Yacht Design
20110601055	Joe	Marker	Computer Science
20130703225	Elena	Gilbert	Public Relations

Table 3.1: `departments` table

StudentID	**RegistrationDate**	**Advisor**
20100703212	27.09.2010	Carrie Coulson
20110501022	24.09.2011	Andrew Garland
20110601055	24.09.2011	Andrew Garland
20120501015	27.09.2012	Matt Donovan

Table 3.2: `advisors` table

Id	**StudentID**	**StudentClub**
1	20080601050	Music Club
2	20130703225	Folk Dance Club
3	20110601055	Aikido Club
4	20120501015	Environmental Club
5	20100703212	Music Club

Table 3.3: `studentclubs` table

Now we will explain relational model, ACID properties, and how data manipulation can be done on relational databases.

3.2.1 Relational Model

Relational model contains structured data and it basically has two main components; relation, and attribute (see Figure 3.2). A *relation* is a set of attributes that define items in relational model.

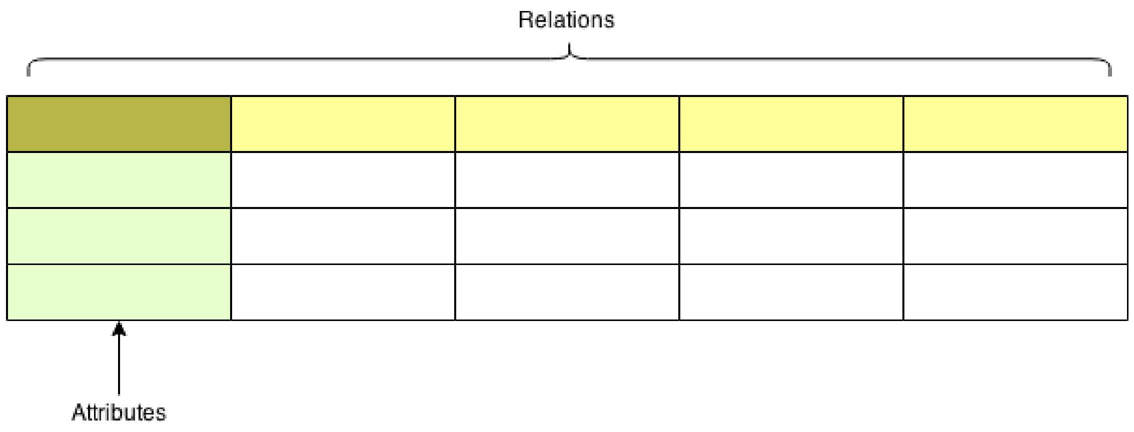


Figure 3.2: Relation and attributes on relational model

- For Table 3.1, relation can be defined as;

  Relation1(StudentID, Name, Surname, Department)

- For Table 3.2 relation can be defined as;

  Relation2(Id, StudentID, RegistrationDate, Advisor)

- For Table 3.3 relation can be defined as;

  Relation3(StudentID, ForeignLanguage, Gpa)

An attribute is a characteristic of an item. Attributes can be referred as the labels of *attribute values*. For example, attributes of Relation1 are *StudentId*, *Name*, *Surname*, and *Department*. Attribute values for Name are *Terry*, *Daniel*, *Daisy*, *Ben*, *Susan*, *Joe*, and *Elena*. Sometimes an attribute value may not be given. In this case **"null"** keyword will be counted as the value for that attribute.

Relation and attribute connections create a table structure. Therefore, finite ordered elements which are created with relations are referred as rows or tuples, and attributes are referred as columns. For example, a tuple in the second row of Table 3.1 is

20100703212	Daniel	Brown	Public Relations

To identify each record uniquely, the user needs to set some attributes (*candidate keys*). One of those attributes will be a *primary key.* This key should always consists of different values for each tuple, meaning that no two tuples can have the same primary key. For example; in Table 3.2, there are registration dates and advisors of students. Sometimes more than one student can register to the same department on the same day. Because of this, their advisor may be the same person. Here we have two tuples sharing same RegistrationDate and Advisor. However, these two records belong to different students, because of the difference in StudentID's. Having a primary key is a must for every database table. It is unique and can include only one attribute, or a group of attributes. Moreover, a *foreign key* represents an attribute or again a group of attributes to create a link between entities in different tables by connecting with the primary key of other table.

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them. For example; in Table 3.3, there are student clubs. Since one student can be registered to more than one student club, StudentID attribute cannot be used as a primary key in this table. Instead, there is another attribute called Id is defined. When there is a need to create a relation with this table and other tables, Id will serve as primary key, and StudentID will be foreign key.

3.2.2 ACID Properties

James Nicholas Gray, was the first person who described *atomicity*, *consistency*, *isolation*, *durability* properties of databases in 1970. Thirteen years later, in 1983, Theo Härder and Andreas Reuter [62] came up with the acronym ACID. Relational databases provide ACID transactions. These properties are very important, because they ensure that database transactions are safe, reliable, and accurate. Next, we provide definitions of these four properties.

3.2.2.1 Atomicity

Any operation performed on the database can be either completely successful or fail. A transaction cannot be separated to be successful, which means that if some fragments of a transaction are successful and the other fragments fail, it cannot be successful at the end; all fragments need to be successful. Therefore, it can be said that each transaction should be atomic.

3.2.2.2 Consistency

Constrains and rules of the database cannot be violated by transactions. Any operation performed on the database should produce consistent outputs. When the same input is given more than once to the database, the results should be the same, unless there were no value changes performed between the first and the last try.

3.2.2.3 Isolation

Transactions should be isolated from each other in any database system. When there are more than one transaction performing concurrent operations, none of them can interrupt or disturb others. For example, if more than one user are changing values of the same item in their own sessions, these changes should remain in their sessions.

3.2.2.4 Durability

Results of transactions should be durable. When an update is committed, the results should stay the same all the time. This allows system recoveries or rollbacks when system fails for some reason during the operation such as CPU, or storage, or a software failures. So that the system will be able to continue working from a restored point.

3.2.3 Normal Forms

Relational database tables may contain in some forms of extra data related to actual data. For example, some attributes or even whole attributes may be duplicated sometimes. This is called *data redundancy*. Relational databases should have as minimum as possible data redundancies. Normalization techniques are introduced to ensure such a requirement. To be able to do normalization, functional dependencies should be provided along with primary keys, and foreign keys of the relations. A *functional dependency* is the basis for normalization. It is a relation between two entities, when one or more attributes identify the value of another attribute uniquely. If there is a functional dependency between relations X and Y, it is signified as $X \rightarrow Y$ and can be read as *"Y functionally depends on X"*. For example, consider table below;

EmployeeID	**Name**	**Surname**	**Profession**	**Salary**
25256	Pepper	White	Software Tester	2500
25645	Peter	Darker	Web Engineer	3000
42545	David	Garner	Web Engineer	3000
63253	Ashley	White	Chief Moderator	3600

Table 3.4: `employees` table

Here, EmployeeID is the determining attribute of Name, Surname, Profession, and Salary attributes. The one who has an EmployeeID of 63253 is Ashley White. Whenever a query with this id number is given as input, the result will be Ashley White. Moreover, if there was another tuple including 63253 as EmployeeId, it would still belong to Ashley White having profession of Chief Moderator, and a salary of $3600. Therefore, it can be said that Name, Surname, Profession, and Salary are depend on EmployeeID.

Functional dependencies are crucial when designing relational database models. By using dependencies, many normalization forms such as first normal form, second normal form, third normal form, Boyce-Codd normal form, and fourth normal form are defined. Now, we are going to explain each.

3.2.3.1 First Normal Form - 1NF

Edgar Frank Codd was the first person who proposed normalization techniques [46] to ensure minimal data redundancies after he introduced relational model in 1971. A relation can be in first normal form if and only if all attributes of a tuple are atomic. In other words, there should not be more than one attribute value in any column. Considering the example below, there are two tables containing Actors, BirthDate, and Movies attributes. In Table 3.5, there are more than one values in Movies column which is not preferred for 1NF. On the other hand, Table 3.6 is in 1NF, since all the values are single in each column.

Actors	**BirthDate**	**Movies**
Rober Downey Jr.	04.04.1965	The Avengers, Iron Man, Sherlock Holmes
Mark Ruffalo	22.11.1967	The Avengers, Just Like Heaven
Chris Hemsworth	11.08.1983	Thor, The Avengers, Rush

Table 3.5: Actors table is not in first normal form

Actors	**BirthDate**	**Movies**
Rober Downey Jr.	04.04.1965	The Avengers
Rober Downey Jr.	04.04.1965	Iron Man
Rober Downey Jr.	04.04.1965	Sherlock Holmes
Mark Ruffalo	22.11.1967	The Avengers
Mark Ruffalo	22.11.1967	Just Like Heaven
Chris Hemsworth	11.08.1983	Thor
Chris Hemsworth	11.08.1983	The Avengers
Chris Hemsworth	11.08.1983	Rush

Table 3.6: Actors table is in first normal form

3.2.3.2 Second Normal Form - 2NF

Edgar Frank Codd [46] proposed another normal form in 1971. A relation can be said in 2NF, whenever it is in 1NF and each attribute that are not included in primary key should depend on all attributes of the primary key. Considering Table 3.7, primary key is Guest, Hotel pair. In this case hotels are not related to the living city of their guests. LivesIn attribute is depending on Guest attribute. Therefore, we need to separate this table into two as in Table3.8. In this form, primary key is Guest and other attributes just depend on this attribute. Second normal can be guaranteed when primary key includes only one attribute.

Guest	**Hotel**	**LivesIn**
Emily Thorn	The Langham, Chicago	Miami
Emily Thorn	Bordessone, California	Miami
Mark Addie	Tivoli Lodge, Colorado	England
Mark Addie	Primland, Virginia	England
Tara Lord	The Oxford Hotel, Oregon	New York
Tara Lord	Shereton, Izmir	New York

Table 3.7: Guest table is not in second normal form

Guest	**Hotel**
Emily Thorn	The Langham, Chicago
Emily Thorn	Bordessone, California
Mark Addie	Tivoli Lodge, Colorado
Mark Addie	French Quarter Inn, Chicago
Tara Lord	The Oxford Hotel, Oregon
Tara Lord	Shereton, Izmir

Guest	**LivesIn**
Emily Thorn	Miami
Mark Addie	England
Tara Lord	New York

Table 3.8: Guest tables are in second normal form

3.2.3.3 Third Normal Form - 3NF

Third normal form is also proposed by Edgar Frank Codd [46]. A relation is in third normal form, if and only if it is in second normal form, and there should not be any transitive dependencies. A *transitive dependency* is a type of functional dependency which contains transitive feature. If attribute Z depends on attribute Y, and attribute Y depends on X, then attribute Z is also depends on X by transitivity. For example, in Table 3.9 an album depends on an artist, and an artist depends on a song. Therefore, an album depends on a song over an artist. This is a transitive dependency and violates 3NF. In Table 3.10, it is shown two separate tables composed from that table and all attributes which are not keys, depend on primary key. In the table on the left, Title depends on Artist, whereas in the other table, Song depends on Artist. Now, these two tables can be said in 3NF.

Album	**Song**	**Artist**
Memories	Memories	Kelly C.
Best Of Kelly	Memories	Kelly C.
Like a Wheel	Alone	Eric Sad
Say My Name	Touch Me	Samantha

Table 3.9: Song table is not in third normal form

Album	**Song**
Memories	Mermories
Best Of Kelly	Memories
Like a Wheel	Alone
Say My Name	Touch Me

Song	**Artist**
Memories	Kelly C.
Alone	Eric Sad
Touch Me	Samantha

Table 3.10: Song tables are in third normal form

3.2.3.4 Boyce-Codd Normal Form - BCNF

Boyce-Codd normal form is an addition to third normal form for a special situation where third normal form does not hold. Therefore BCNF is a stronger version of 3NF. It is proposed by Raymond F. Boyce and Edgar Frank Codd [47] in 1974. A relation is in BCNF, if and only if each and every determinant is a candidate key. This means all non-trivial functional dependencies have super keys on the left hand side. We do not provide any examples here.

3.2.3.5 Fourth Normal Form - 4NF

BCNF is a common used normal form. However, it can still have some anomalies. To overcome these issues, fourth normal form is introduced. 4NF is proposed by Ronald Fagin [57] in 1977. 1NF, 2NF, 3NF, and BCNF are related with functional dependencies. On the other hand, 4NF is provided by multivalued dependencies. A *multivalued dependency* appears in the same table when there are one or more tuples contains one or more other tuples. It can be signified as $X \rightarrow\rightarrow Y$. For example, in Table 3.11, there are car models, companies, and color of the cars. A company can produce multiple colors of the same model. Therefore, CarModel and Company values are duplicated when Color changes. There are multivalued dependencies between CarModel and Company attributes, as well as CarModel and Color attributes, so that primary key is the pair of CarModel, Company attributes and this causes violation of 4NF. Consequently, the table is decomposed into two as in Table 3.12 to eliminate unnecessary duplications.

CarModel	**Company**	**Color**
M1	BMW	Black
M2	BMW	Red
M2	BMW	Blue
M3	BMW	Red
M3	BMW	Black
A3	Audi	Black
A3	Audi	White

Table 3.11: Car table is not in fourth normal form

CarModel	**Company**
M1	BMW
M2	BMW
M3	BMW
A3	Audi

CarModel	**Color**
M1	Black
M2	Red
M2	Blue
M3	Red
M3	Black
A3	Black
A3	White

Table 3.12: Car tables are in fourth normal form

3.2.3.6 Other Normal Forms

There are also fifth normal form (5NF or PJNF), sixth normal form (6NF), and seven normal form (7NF) in the literature. However, they are out of the scope of this thesis.

3.2.4 SQL: Structured Query Language

3.2.4.1 Definition

Data is manipulated by using *Structured Query Language* (SQL) in relational databases. It is first designed to support quasi-relational database management system at IBM in 1970 by Donald D. Chamberlin and Raymond F. Boyce [42]. They named it as SEQUEL which stands for Structured English Query Language. Later on, the name changed into just SQL. In 1979, SQL was released commercially for the first time ever to the market.

3.2.4.2 Retrieving Data

SQL is based on queries, and consists three main clauses; `SELECT` ,`FROM`, and `WHERE` , in order to **retrieve** data from a single table or a set of tables.

- `SELECT` clause describes the attributes to be retrieved as a result of the query.
- `FROM` clause indicates the tables which are required to read for the query.
- `WHERE` clause restricts and filters query results by using an equation or a combination of equations.

Let us write a very simple SQL query to find *"the advisor of student whose id is 20120501015"* in Table 3.2. Firstly we need to select advisors, therefore we need a `SELECT Advisor` statement first. We use `advisors` table, therefore we need a `FROM advisors` statement. Lastly, we need to find student whose id is *20120501015*, therefore we need a `StudentID = 20120501015` statement. The query would then return the following result:

Advisor
Matt Donovan

The order of the statements in an actual query format is shown below;

```sql
SELECT Advisor
FROM advisors
WHERE StudentID = 20120501015;
```

Of course, queries not always that simple. Most of the time there are more than one tables contain related information with each other. In order to bring tables together and search for information, **join** procedures should be performed. The most common way of joining tables is done by using equal (=) symbol in the `WHERE` clause.

Let us write another query to find *"name and surname of a student whose advisor is Andrew Garland, and whose department is Computer Science"*.

```
SELECT Name, Surname
FROM departments, advisors
WHERE departments.StudentID = advisors.StudentID AND
      advisors.Advisor = "Andrew Garland" AND
      departments.Department = "Computer Science";
```

Two tables `departments` and `advisors` join together, according to the corresponding *Name* and *Surname* attributes of *Avdisor* Andrew Garland and *Department* Computer Science. `AND` allows combining conditions. Here, we have three conditions; joining tables, finding advisor, and finding department. Therefore, these conditions are combined together with `AND`. SQL also provides `NOT` to negate a given condition, and `OR` to check both conditions and return satisfying one. The query above returns this result:

Name	**Surname**
Joe	Marker

There are different types of join operations such as *inner join*, *left join*, *right join*, *full join*, *cartesian join*.

- **Inner Join:** If there is a match in both joining tables, returns all rows.
- **Left Join:** Returns all rows of left table, although there are no matching rows at the right table.
- **Right Join:** Returns all rows of right table, although there are no matching rows at the left table.

- **Full Join:** If there is a match in one of joining tables, returns all rows.
- **Cartesian Join:** Returns Cartesian product of rows from joined tables.

SQL provides additional statements to analyse further and display data. These are `GROUP BY`, `HAVING`, and `ORDER BY` clauses.

- `GROUP BY`: This clause groups retrieved data with respect to the given conditions.
- `HAVING`: This clause filters retrieved data with respect to the given conditions. Filtering operation is similar to `WHERE` clause, however `HAVING` filters grouped results.
- `ORDER BY`: This clause sorts result set with respect to the given conditions including ascending (`ASC`), and descending (`DESC`) order options.

Aggregation of data is also possible with SQL aggregating functions; `COUNT`, `AVG`, `SUM`, `MIN`, `MAX`.

- `COUNT`: This function counts the number of rows with respect to the given conditions.
- `AVG`: This function finds average value of a column with respect to the given conditions.
- `SUM`: This function calculates sum of a column with respect to the given conditions.
- `MIN`: This function finds lowest value of a column with respect to the given conditions
- `MAX`: This function finds highest value of a column with respect to the given conditions

3.2.4.3 Adding, Modifying, and Removing Data

SQL uses `INSERT`, `UPDATE`, and `DELETE` statements, in order to **write** data to a single table, or a set of tables.

`INSERT` clause is used to add a new data to existing tables. For example, if we want to add a new student record (whose StudentID is 20140401040, and registered to Aikido club) onto Table 3.3, we have to execute the following query:

```sql
INSERT INTO studentclubs(Id, StudentID, StudentClub)
VALUES (6, 20140401040, "Aikido Club");
```

A new row will be created and added to the end of the table as follows:

Id	**StudentID**	**StudentClub**
1	20080601050	Music Club
2	20130703225	Folk Dance Club
3	20110601055	Aikido Club
4	20120902015	Environmental Club
5	20100703212	Music Club
6	20140401040	Aikido Club

Table 3.13: `studentclubs` table

`UPDATE` clause is used to change the values of existing tables. For example, if we want to modify the salary of an employee (whose EmployeeID is 25256) on Table 3.4, we have to execute the following query:

```sql
UPDATE employees
SET salary = 3200
WHERE EmployeeID = 25256;
```

Column value of the row which belongs to Pepper White will be changed and the table will be modified as follows:

EmployeeID	Name	Surname	Profession	Salary
25256	Pepper	White	Software Tester	3200
25645	Peter	Darker	Web Engineer	3000
42545	David	Garner	Web Engineer	3000
63253	Ashley	White	Chief Moderator	3600

Table 3.14: `employees` table

`DELETE` clause is used to delete rows from existing tables. For example, if we want to delete an employee (whose name is David, and surname is Garner) from Table 3.14, we have to execute the following query:

```
  DELETE FROM employees
  WHERE Name = "David" AND Surname = "Garner";
```

The entire row which belongs to David Garner will be deleted and the information will remain as follows:

EmployeeID	Name	Surname	Profession	Salary
25256	Pepper	White	Software Tester	2500
25645	Peter	Darker	Web Engineer	3000
63253	Ashley	White	Chief Moderator	3600

Table 3.15: `employees` table

3.3 NoSQL: Not Only SQL

Although relational database management systems are dominating others for more than forty years, technological methods need to shift, due to find solutions to new requirements. We discussed in Chapter 2 what Big Data is and why traditional database management systems do not work well with it. In addition, we discussed what MapReduce and Hadoop are and how they work. Since there is not a query language such as SQL in the structure of Hadoop, it acts more like a warehouse rather than a database. Thus, NoSQL term comes into play. It is a database technology which aims to process the data which is high in volume, includes different sorts of sources, and generated very fast.

3.3.1 Definition

The term NoSQL first came out in 1998 by Carlo Strozzi [82]. At that time he used this term to provide no form of the SQL for querying. In late 2009, Eric Evans from Rockspace, a cloud hosting company, reused NoSQL word in a conference to address a different meaning; "Not only Structured Query Language".

"NoSQL systems are distributed, non-relational databases designed for large-scale data storage and for massively-parallel data processing across a large number of commodity servers." (Moniruzzaman and Hossain, 2013) [76]. Since SQL does not meet the needs of big online companies who produces mass amount of data every hour such as Amazon, Google, Facebook, and Twitter any more, NoSQL started to be used and improve their performance.

3.3.2 BASE Properties

Database transactions should be done considering ACID properties to ensure reliability. They are working well with relational models which are depending on schemas, and structured data. However, NoSQL models sometimes violate one or more of these properties depending on the design. Parallel programming, distributed structure, and unstructured data are no longer maintain the stability of ACID.

Eric Brewer presented CAP theorem during ACM Symposium in 2000 [40]. Distributed systems need to ensure *consistency*, *availability*, *partition tolerance* to create stable designs. Yet, none of them can provide all three properties at the same time. For example, distributed systems should provide availability and partition tolerance at any time, on the other hand, this fact reduces consistency. As a counter part of ACID, Eric Brewer introduced BASE properties.

3.3.2.1 Basically Available

The system does guarantee availability with respect to CAP theorem. There should be a response to any request and this may be either a success, or a failure.

3.3.2.2 Soft-State

The system can change its state over time. Sometimes there may be no inputs given, but the state can still change for consistency.

3.3.2.3 Eventual consistency

Over time, the system eventually becomes consistent when incoming inputs are no longer exist.

3.3.3 NoSQL Models

There are four types of NoSQL database models. Even though they are all aimed to behave as a database for a system, they do the job in different ways. Next, we explain those models briefly except graph databases.

3.3.3.1 Key-Value Stores

Key-value stores model is based on Amazon's article (2007) [51] about their distributed data store named Dynamo. It is the simplest category of NoSQL and relies on a global collection of key-value pairs. Basically it is a schema-less model that is consist of a numerical identifier to represent the key and the actual data in the format of strings, sets, or even lists to represent the corresponding values. This type of a structure scales itself as a hash table as the data increases. More importantly, querying should be carried out through keys not values.

Key	**Values**
1	Password: xxx Name: Craig
2	ID: 239344 Location: Izmir Occupation: Engineer
3	Last Name: Yates

Table 3.16: A key-value store example

Table 3.16 is a representation of a key-value store. The data is unstructured because vales are not fixed as they are in relational models. There may be one or more values with different types of attributes for associated key. Generally key-value stores are used when the data is highly scalable and fetching is intended to be really fast. They are used in applications to manage a user's profile, information of a session, finding items associated to a certain key value, and etc.

Some examples of key-value stores are:

- Redis [24] (an open-source product, copyrighted by Salvatore Sanfilippo and Pieter Noordhuis in 2009)
- Dynamo [11] (offered by Amazon in 2007)
- Riak [25] (an open-source product, developed at Basho Technologies in 2009)
- Voldemort [30] (offered by LinkedIn in 2009)
- Windows Azure Storage [31] (offered by Microsoft in 2008)
- Aerospike [2] (offered by Aerospike in 2012)

3.3.3.1.1 Key-Value Stores Applications

Key-Value stores are simple databases to build, and manage the data. One of the well-known example is session data of the users. By using id of a user, it is possible to retrieve and update relative the data from the store. Another example is accessing the most current messages of a user or favourite items. The system should know the keys of messages or items and recompute information on the background regularly to get current status of them.

3.3.3.2 Column-Family/Wide-Column Stores

Column-family/wide-column model is based on Google's article (2006) [43] called Bigtable. In the column family object, there are columns of related data. In fact key-value pairs still exist in column family type, but they are mapped to the set of columns. Columns can be modelled as tables and key-value pairs can be modelled as rows in a relational database model.

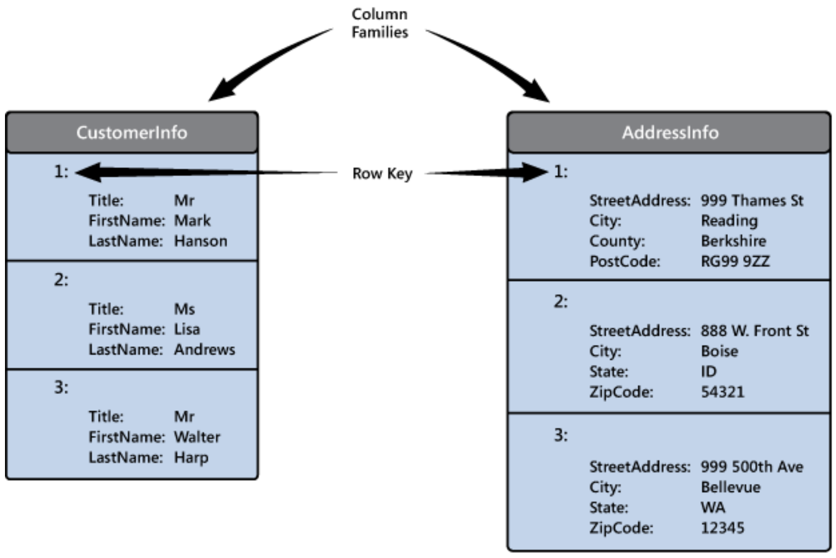


Figure 3.3: A column-family/wide-column store example [10]

Figure 3.3 shows the representation of a column family/wide-column store. Here CustomerInfo and AddressInfo are column families.

Moniruzzaman and Hossain (2013) [76] states that using column family/wide-column stores are best for:

- Distributed data storage,
- Large-scale, batch-oriented data processing,

- Exploratory and predictive analysis performed by expert statisticians and programmers.

Some examples of column-family/wide-column Stores are:

- Bigtable [43] (offered by Google in 2006)
- Hbase [13] (developed by Apache)
- Cassandra [7] (an open source product developed at Facebook)
- Hypertable [15] (an open source product developed at Zvents Inc.)
- Accumulo [1] (created by National Security Agency)
- SimpleDB [26] (offered by Amazon)

3.3.3.2.1 Column-Family/Wide-Column Stores Applications

Column-family/wide-column stores are more complex when compared to key-value stores. This type of databases are beneficial when column operations such as MIN, MAX, and AVG will be used frequently. When there are many transactions going on with the application, this type of a database should not be used. Because, delete and update operations will be reducing storage efficiency.

3.3.3.3 Document Databases

Document databases are originated from IBM's Lotus Notes software. They are designed to store and manage semi-structured data which means neither raw nor typed data. It is actually structured but not organized as a relational structure. They have the ability to store dynamic documents in a database. Therefore a client is able to edit the existing documents. Document encoding can be done with different formats e.g. XML (Extensible Markup Language), JSON (JavaScript

Object Notation), BSON (binary encoding of JSON objects), YAML (Yet Another Markup Language), or PDF (Portable Document Format).

Document oriented databases can be modelled as rows in a relational database model, but their structure is based on key-value pairs rather than a table structure. Querying can be performed not only with keys but also with values.

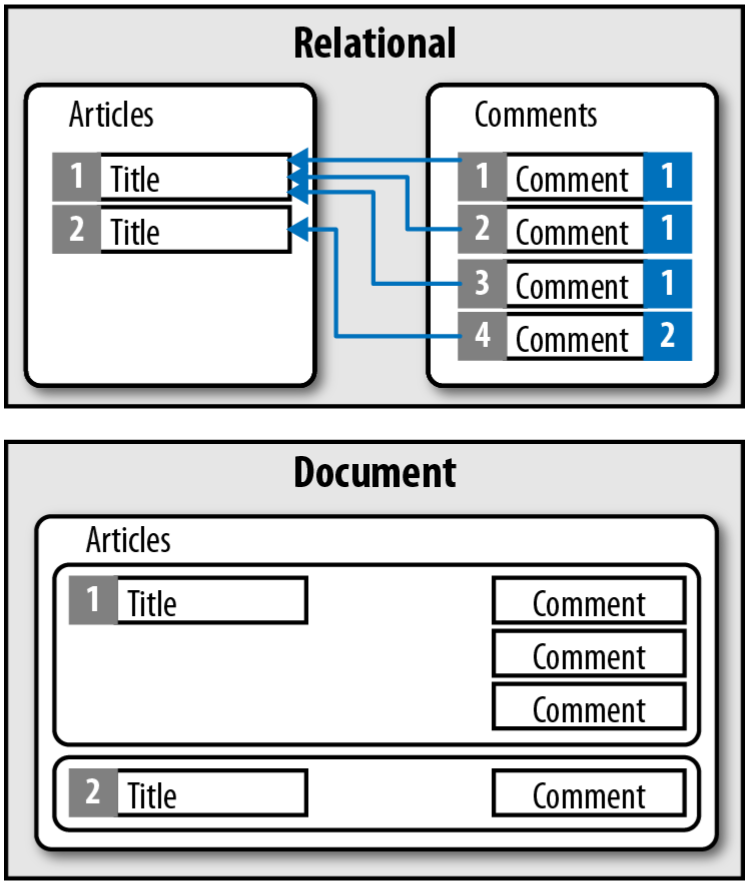


Figure 3.4: Relational model vs document model [73]

Figure 3.4 shows how a document database model differs from a relational database model. For this example comments which related to an article is kept together to form a document.

Some examples of document databases are:

- MongoDB [18] (an open source application developed by 10gen in 2009)

- CouchDB [8] (an open source application developed by Damien Katz in 2005)
- RavenDB [23] (developed by Ayende Rahien in 2009)
- Elasticsearch [12] (an open source application developed by Shay Banon in 2010)
- ThruDB [27] (developed by T Jake Luciani in 2007)
- JasDB [17] (offered by Obera Software in 2012)

3.3.3.3.1 Document Databases Applications

Document databases are good when the aim is to store data as documents with certain characteristics and also if other documents are continuously added to the system, getting updates or completely removed. The application will require a CRUD (create, read, update, and delete) user interface such as a simple contact list of a mobile phone. Hecht and Jablonski (2011) [65] indicates other examples such as blogs and real-time analytics.

3.3.3.4 Graph Databases

Graph database idea comes from the graph theory. Buerly (2012) [41] explains that the graph theory is emerged in 1736 when Euler solved the Seven Bridges of Königsberg problem by modeling a graph topology with nodes and their connectors. By using this model he answered that he can walk through the city by crossing each bridge only once. After that, graph theory became very popular and used for many well-known problems such as minimum spanning tree (Borůvka, 1926) [39], graph enumeration (Harary and Palmer, 1973) [63], clique problem (Luce and Perry, 1949) [72], and etc. And nowadays it is proved that it can be used to model database management systems too.

Graph databases are highly aimed to store information about networks. They include nodes to represent entities, edges to represent the relationships, and properties to represent information about these nodes and edges. They are generally built and optimized for traversing over the graph and matching patterns if exist, producing recommendations for social networks, and etc.

Urma and Mycroft (2013) [84] states that graph databases are differing from relational database models with two fundamental properties; index-free adjacency, and semi structured data. Graph databases support index-free adjacency; meaning that each node knows the location of its adjacent nodes. Thus an index lookup becomes unnecessary. The data is gathered by accumulating the information of nodes and their links. This allows accessing results faster than relational database models. Graph databases also support semi-structured data explained in document databases section. Additionally join operations are unnecessary for graph databases because a graph scales itself naturally as data increases.

Some examples of graph databases are:

- **Neo4j** is developed by Neo Technology in 2007 [19]. It is the most popular graph database product and a well documented open source project. It is built-in Java and fully transactional. Community edition and Personal version of Commercial subscriptions are free, whereas Start-ups and Business & Enterprise editions requires certain amount of fees. It is cross platform and working on Linux, Mac, and Windows. Other than Java, Neo4j has many drivers and supports .net, JavaScript, Ruby, Python, PHP, Scala, and Clojure which makes developers flexible to play with it. With the release of version 2.0, it has a user interface too. It is very easy to install and connect by just selecting folder path from your files and works through your browser with REST API. There are nodes and relationships between them. There is not any limit on the number of nodes. Query language of Neo4j is called Cypher. Both nodes and relationships can now be labelled. This makes searching faster and improves indexing when writing queries.

- **AllegroGraph** is offered by Franz Inc. [3] It is initially aimed to support Resource Description Framework. It is cross platform and working on

Linux, Mac, and Windows. It has client interfaces (similar to Neo4j) for Java, Python, Ruby, Perl, C#, Clojure, and Common Lisp. It uses Gruff: A Grapher-Based Triple-Store Browser for graphical query builder. The most important usage of AllegroGraph is RDF triples.

- **Trinity** is offered by Microsoft in 2013 [29]. It uses a memory cloud so that it is a memory-based graph database supporting low-latency online query processing. It supports only C# APIs for now. It can run in two modes; embedded and distributed. It is not open source but still in development.

- **InfoGrid** is registered to NetMesh Inc. since 2010 [16]. It is described as a Web Graph Database aimed to make web applications on a graph system easily. It is an open source project and still being in development with different projects such as InfoGrid Model Library Project, InfoGrid Probe Project, and etc.

- **BrighstarDB** is developed by Khalil Ahmed and Graham Moore [6]. It is scalable graph database for .NET platform and can be used as an Azure service. Its query language is LINQ. Its difference from other products comes from providing an Entity Framework model for data storage.

- **TitanDB** is a scalable distributed graph database offered by Aurelius in 2012 [28]. It is build on Apache Cassandra, open source and supports billions of nodes. There are three main storage back ends; Cassandra, HBase, and Oracle Berkley. In order to make a search, it should be first indexed. It uses a native integration with TinkerPop graph stack. Its query language is Gremlin, uses Frames as object-to-object mapper, and graph server is Rexter.

3.3.3.4.1 Graph Databases Applications

Graph databases work best when a social networking is large part of the application such as Facebook and Instagram. Because Graph databases rely on traversing over linked data. You can look for what people are liking, buying, posting. After gathering these informations, you can create algorithms to recommend pages to users related to their activities that have been done before. You can create queries and ask for specific records such as *"which of my friends like product A but not product B and have never been to city C?"*. Graph databases are strongly related to the aim of our project and its working principles. Therefore, we chose to develop a graph database to compare graph design with the relational model in Chapter 5.

3.3.3.4.2 Graph Model

Graphs are composed of two main components; vertices **(V)**, and edges **(E)**. A *vertex* is used to represent an object within the context of the model and sometimes referred as *node*. An *edge* is used to represent connections between vertices. Thus, they are used to create relationships among nodes. This structure together is called as a *graph* **(G=(V, E))**. In order to connect two nodes, one edge should be used. Directions of edges can be specified, therefore that kind of graph will be a *directed graph* (see Figure 3.5). If edge directions are not specified, then this graph referred as an *undirected graph* (see Figure 3.5). From node A to node B, and from node B to node A will be the same edge.

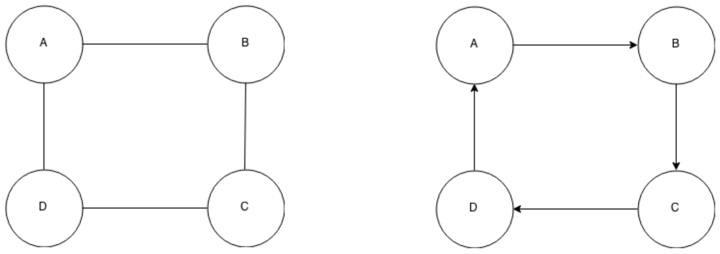


Figure 3.5: An undirected graph (right), and a directed graph (left)

Graphs on Figure 3.5 are *labelled graphs*, because nodes are labelled as *A*, *B*, *C*, and *D*. A *path* from node *A* to node *D* is shown with bold edges below:

Consider this scenario; Tony is a friend of both Henry and Kate. He works at Oxford and Kate owns a Mercedes. Figure 3.7 is a representation of a graph database resulted from previous scenario. It can be understood from the figure (on the left) that graph databases are really whiteboard friendly. It is easy to

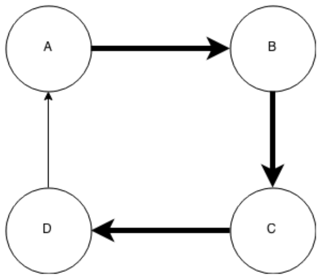


Figure 3.6: A path from A to D

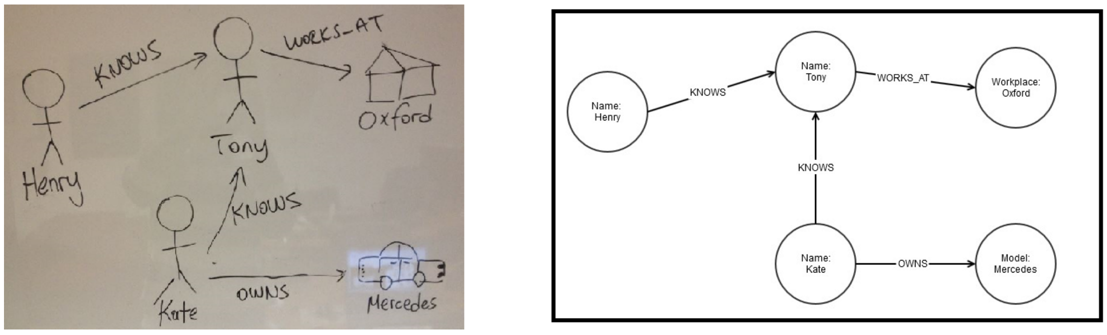


Figure 3.7: A graph database example

model a graph database on whiteboard by drawing initial ideas quickly. Then as the time passes and the model can be improved easily by drawn modifications and transfer to the computer (on the right) before starting to code the database.

One or more relational tables can be merged into a single graph in graph database design. Let us combine Table 3.1, 3.2, and 3.3 and design a graph database.

First, we need to decide nodes and their types. *StudentID*, *Name*, and *Surname* attributes are related to a student object in Table 3.1. However, *Department* is not dependent on students, therefore it can be separated from students and become another object itself. Since *Advisor* attribute in Table 3.2 is also not dependent on students, it can be separated too. In Table 3.3, *SudentClub* is an object in real world. Hence, we have 4 types of nodes; **Student**, **Department**, **Advisor**, and **StudentClub**.

Now, we need to create relationships and their directions between nodes. A student studies at a department, therefore the relationship from a student node to a department node can be *study_at*. A student may be joined to a club, therefore the relationship from a student node to a student club node can be *joined*. An advisor coaches or guides a student, therefore the relationship from a student node to an advisor node can be *guided_by*. Hence, we have 3 types of relationships; STUDY_AT, JOINED, and GUIDED_BY.

Finally, our graph database will look as in Figure 3.8.

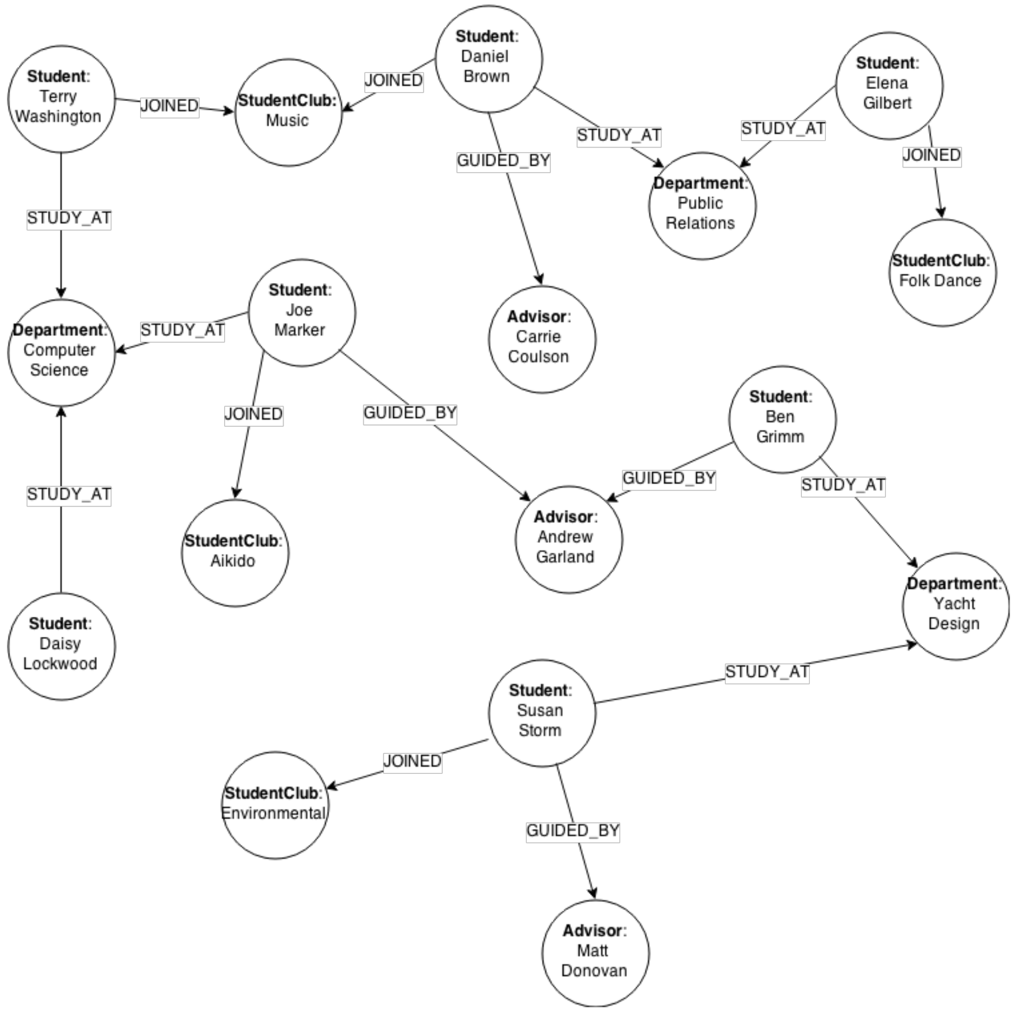


Figure 3.8: Student graph database

There are various graph database products in the market as we listed before. We decided to design our graph database using Neo4j in Chapter 5. Because it is free, easy to install, well-documented, portable, mature, and supported. Cypher is easy to learn, similar to SQL. It meets all needs of TrendPin when it comes to huge amount of data management. We explain how Cypher processes data in the next section.

3.3.3.4.3 Cypher Query Language

Although there are different relational database management systems, most popularly MySQL, Oracle, DB2, Access, and MSSQL, SQL became a standard query language for those and more products. As for graph databases, there are various query languages such as Cypher and Gremlin. Since Neo4j works with Cypher, we discuss this language by providing example queries on various graph models.

Cypher developers tried to create an easy to read query language which is enriched with mimics of nodes and vertices of a graph. A simple Cypher query is composed of `MATCH`, `WHERE`, and `RETURN` clauses in order to retrieve data from graphs.

- `MATCH`: This clause specifies a searching example or pattern on the graph. It is an illustration of the searched information. It uses left and right parentheses (`()`) to indicate nodes, dashes (−) to indicate relationships, left and right angle brackets (< >) to indicate the direction of relationships, square brackets (`[]`) and colons (`:`) to indicate relationship names.
- `WHERE`: This clause is similar to SQL's `WHERE` clause. It filters matching results with respect to a given criteria.
- `RETURN`: This clause defines nodes, relationships, or properties to be displayed to the user.

Let us write a simple Cypher query below for the graph on Figure 3.7 to *find the name of the person who works at Oxford*. We first need to find the node which represents *Oxford*, then we need to filter the relationship labelled with `WORKS_AT` between *Oxford* and the person we are looking for. Finally, we need to return the name of the corresponding person.

```
MATCH (u) -[:WORKS_AT]-> (p)
WHERE p.name = "Oxford"
RETURN u.name
```

As a result, it returns one record which is *Tony*. Person node is represented with u, and workplace node is represented with p. Relationship between these two nodes is represented with `WORKS_AT` label. `WHERE` statement filters workplaces whose name is *Oxford* and associates it with node p. `RETURN` statement chooses only *name* attribute of people found and displays to the user.

As for another example, let us *find who owns a Mercedes*. We first find the node which represents *Mercedes*, then we filter `OWNS` relationship with *Mercedes* and the person. However this time, we are going to change the direction of the relationship, in order to prove that direction is related with the node we are looking for in the graph. Then we return the name of the corresponding person. The result will be *Kate*, and the query below does returns it.

```
MATCH (c) <-[:OWNS]- (u)
WHERE c.model = "Mercedes"
RETURN u.name
```

In graph databases, there is no need to join tables as in relational databases. Everything is encapsulated on the graph. If there is a node with no connection to the graph, it can easily be connected by just creating a connection. Cypher provides nearly all aspects of SQL and more, in terms of data manipulation. Graph traversals are easy with Cypher. There is even a function to find shortest path `shortestPath()`. On the other hand, Cypher supports the followings:

- Sorting items with `ORDER BY`
- Counting the number of records with `COUNT`
- Returning distinct results with `DISTINCT`
- Limiting the number of records with `LIMIT`

- Aggregating values in the result set with `SUM`
- Creating chains among query results with `WITH`
- Allowing indexes with `USING INDEX`,

In Neo4j, nodes can be labelled. The advantage of labelling is to increase search speed by restricting search conditions on requested labelled nodes, rather than searching the whole graph. Have a look at the students graph in Figure 3.8 and *find how many students are guided by an advisor named Andrew Garland*. If the nodes are not labelled, the query would be scanned all over the graph. However, this time only the nodes labelled with `student`, and `advisor` will be enough to get the correct results. Since the question asks to find how many of them, we use `COUNT` clause in `RETURN`. There are two students who are guided by Andrew Garland on the graph, hence, the result of the query bellow will be *2*.

```
MATCH (s:Student) - [:GUIDED_BY]-> (a:Advisor)
WHERE a.Name = "Andrew Garland"
RETURN COUNT(s)
```

Cypher uses `CREATE`, `SET`, `DELETE`, and `REMOVE` statements to do modifications on a graph.

- `CREATE`: This clause is used to create new nodes, as well as new relationships. Creating a new node requires a node identifier, a label, keys and values inside curly brackets (). On the other hand, creating a new relationship requires node mimics before and after the relationship, and optionally directions.
- `SET`: This clause is used to define new values and do the change on nodes or relationships.
- `DELETE`: This clause is used to delete existing nodes or relationships.
- `REMOVE`: This clause is used to remove labels or properties of nodes or relationships.

In order to add new data on a graph, first new nodes should be created. Basically just using `CREATE` clause and its requirements are enough to create new nodes. Let us add a new node to student graph we illustrated in Figure 3.8 using the query below:

```
CREATE (s:Student {StudentID: "20130601010",
                   Name: "James",
                   Surname: "Rhodey",
                   RegistrationDate: "26.09.2013"})
```

In this query, `s` is the node identifier, `student` is the label, `StudentID`, `Name`, `Surname`, and `RegistrationDate` are keys, `20130601010`, `James`, `Rhodey`, and `26.09.2013` are values.

Creating a node does not mean that it will be connected to the graph immediately. To create a relationship, first `MATCH` clause is required to find the nodes which are going to connect each other. Additionally `WHERE` clause helps to match nodes with given properties. Then, `CREATE` is used to create the relationship. Considering student graph on Figure 3.8 again, let us say that *James is registered to Aikido club* and we want to add this node to a student club node. We use the following query:

```
MATCH (s:Student), (c:Club)
WHERE s.StudentID = "20130601010" AND
      c.Name = "Aikido Club"
CREATE (s) -[:JOINED{ id = "5"}]-> (c)
```

Updating a property of a node can be done by `SET` clause. To do this, first `MATCH`, and `WHERE` clauses are again required to find the nodes which will be affected from updates. Then, `SET` clause updates selected values. Considering Figure 3.8, Elena Gilbert, whose student id is *20130703225*, wants to change her department from *Public Relations* to *Public Relations*. In this situation, both her department and her *StudentID* need to be updated. The query below provides these updates:

```
MATCH (s:Student)
WHERE s.StudentID = "20130703225"
      s.Name = "Elena Gilbert"
SET s.StudentID = "20130103225" AND
     s.Department = "Computer Science"
```

Deleting a node or a relationship is provided by `DELETE` clause. Consider Figure 3.8 once more and assume that a student, whose name is *Susan Storm*, left from *Envionmental* club. Firstly, `MATCH` clause needs to scan students who are joined student clubs by representing `JOINED` relation with `r`. Secondly, `WHERE` clause narrows these results to the *Susan Storm* and *Environmental*. Finally, relation `r` gets deleted as follows:

```
MATCH (c:Club) <-[r:JOINED]- (s:Student)
WHERE s.Name = "Susan Storm" AND
         c.Name = "Environmental"
DELETE r
```

3.4 Summary

In the relational databases, the information is kept in the form of tables and attributes. Information is then retrieved from many tables by join operations. It is also possible to use constraints on the query that can be used due to the qualification of data that is being searched for. Atzeni et. al. (2012) [38] claims that relational database models are used for structured data more than 30 years. Technology is not stable. It is changing very fast. During all those years relational database models have been compared with new ideas such as object oriented databases. With the introduction of NoSQL, researchers, developers, and scientists are started to compare relational database systems and NoSQL systems. According to a White Paper from Datastax (2013) [49], technology leaders are now looking forward to directly move to NoSQL systems rather than just considering to have a NoSQL based system or not.

As opposed to the table structure of relational database models, NoSQL has no tables. Instead, there are key-value stores. The purpose of these stores is to extract data easily, in a simple format, and manage it to increase the efficiency while avoiding the problems caused by the latency of data generation. Data is stored both in a JSON or XML document format. Therefore, there is no join operations, and no complex queries are needed.

ACID, properties are the core attributes of relational database models. In order to improve performance and scalability, Nance et. al. (2013) [77] states that NoSQL database systems are not usually being tied to these attributes. Thus, another term called BASE has been emerged. On the other hand Singh (2013) [80] states that although NoSQL data models do not usually support ACID attributes of relational databases for scalability reasons a new platform called FoundationDB is now able to combine ACID and NoSQL transactions well enough.

Graph databases provide easy data modelling as graphs on the board before even writing any codes. By using this advantage, they became popular among other NoSQL models. They are very suitable when modelling social graphs including many relationships among thousands of objects. Horizontal scaling is another advantage of graph models. Traversals are faster with provided functions even depth of the paths are huge such as "friends of friends of friends" relationships.

There are various NoSQL models, which we have not discussed in detail. However, we chose graph databases in addition to relational model for this thesis project, therefore we explained graph model in detail. In the next section, we present TrendPin as a Big Data project.

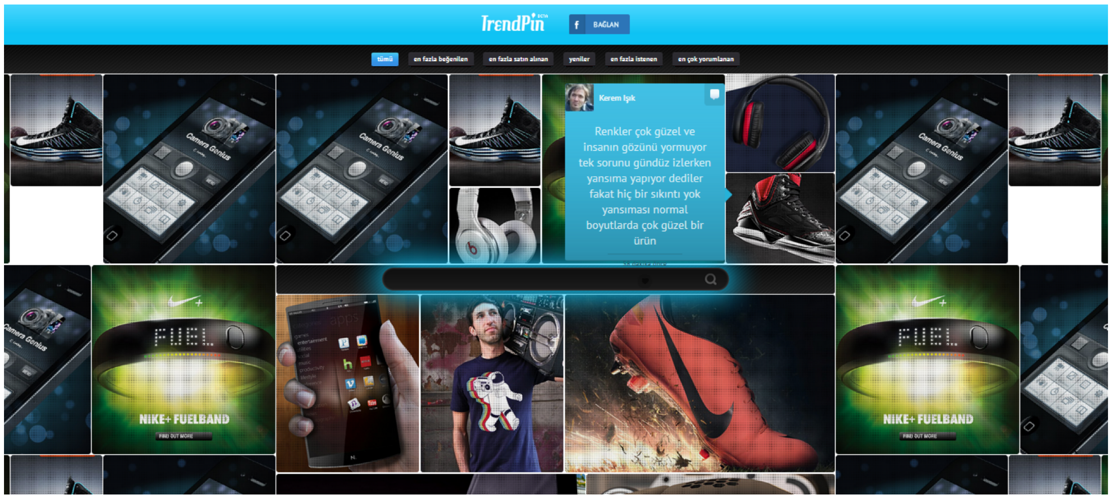
Chapter 4

TrendPin

4.1 Introduction

Online shopping is a popular platform for many people, who like to buy products or services through the Internet using technological devices such as computers, tablets, or applications on smart phones. There are a lot of advantages of this kind of shopping. For example, one can check e-shops quickly for specific items at home, at job, or even walking outside, since they are available 24/7. Price comparisons are also very easy by just typing the name of the item on different online stores and clicking the item links. Therefore, there is no need to run from one shop to another and check those items. If a digital media has been purchased, it can immediately become available to download. On the other hand, if a product has been purchased, package can be delivered to anywhere the user wishes. Of course, there are some disadvantages too. For instance, payments are usually being done by credit cards, cheques, or pre-paid cards. However, some e-shops provide cash on delivery options too. Privacy and security are also problematic. Some users do not want to share personal information with these kinds of providers.

TrendPin is a social network founded by Murat Demir and Kerem Işık in 2012. It is an online shopping system which completely redefines this experience by adding a social dimension on it. It aims to reduce doubts when buying a product with special features to decide whether it is good, bad, worth buying, or even allows you to see if it has already been bought by your friends. These features makes TrendPin the best platform to make people better choices when doing shopping on the internet.



Kerem Işık

Renkler çok güzel ve insanın gözünü yormuyor tek sorunu gündüz izlerken yansıma yapıyor dediler fakat hiç bir sıkıntı yok yansıması normal boyutlarda çok güzel bir ürün

Figure 4.1: Homepage of TrendPin

4.2 Features

There are mainly two features of TrendPin; online shopping and social networking. You can think something like Amazon meets Facebook at first. However, TrendPin is far more beyond that mentality.

First of all, users have to connect with their Facebook account and invite their Facebook friends to TrendPin in the first version of the system. There are user profile pages and product pages in TrendPin. After they personalized their profile page, they can search for a product which they wish to buy or just get to know more about it. On a product profile page (see Figure 4.2), there is a picture (see Figure 4.2–a), name (see Figure 4.2–b), and description (see Figure 4.2–d) of the product, as well as the prices of the same product for different firms (see Figure 4.2–f). When the user clicks to the links of the firms, they can be redirected to the product page on their website too.

4.2.1 Core Features

There are three buttons (see Figure 4.2–c) which are labelled as;

- **Like**
- **Have**
- **Buy**

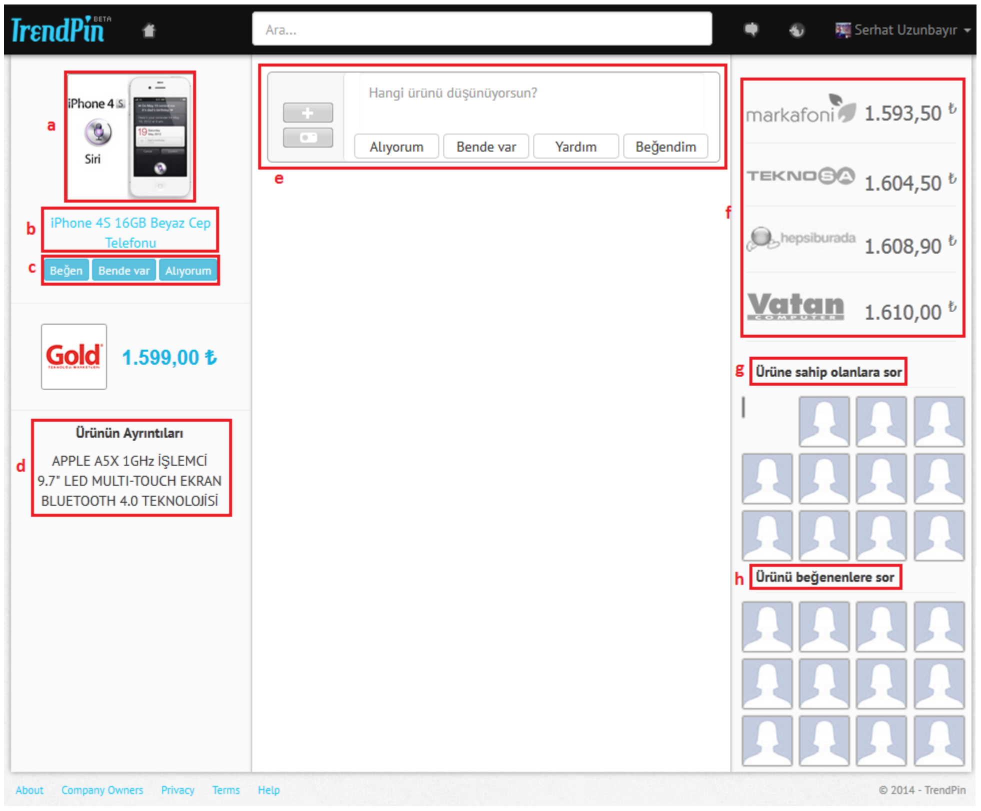


Figure 4.2: Profile page of a product

These three buttons are essential features for the core idea of TrendPin. When the users click one or more of these buttons, it will be shown on their profile page in a box (see Figure 4.3) to show however they indicated that product; either as they like or have, and their picture will be shown on that product's page as well. The reason behind this idea is that if the friends of users want to buy a product that these users already have, friends of them can see their picture on the product's page, and can ask questions and opinions about it easily by just clicking to the users picture (see Figure 4.2–g and 4.2–h).

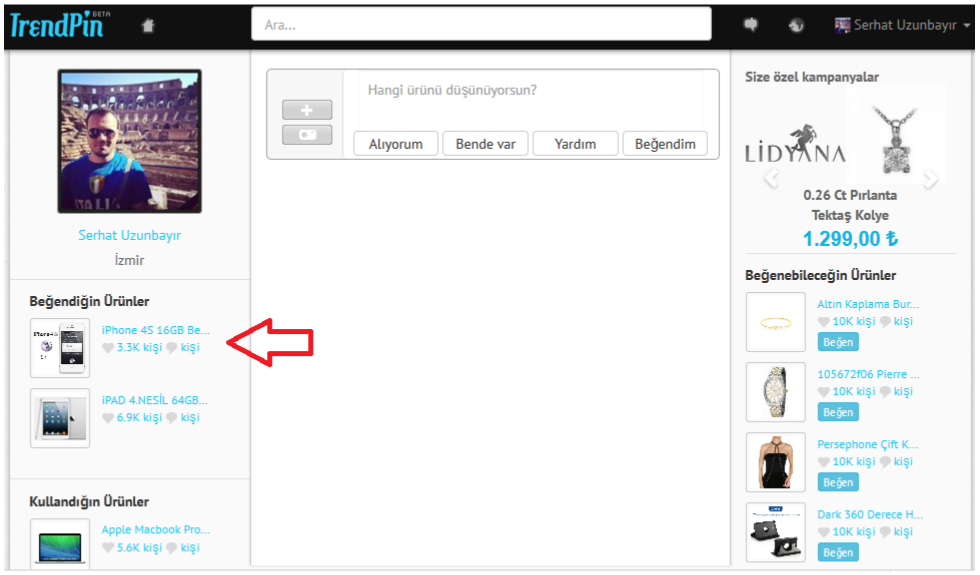


Figure 4.3: Profile page of a user

As in the most online shopping applications, the user can share their comments, add pictures and videos, share links, and etc. about the product on its profile page, or their own profile page using comment text field (see Figure 4.2–e) by adding a mention character right before the name of the product. In TrendPin the users have to use plus sign (+) in order to mention a product. Here, another important feature of TrendPin should be underlined. On Figure 4.2–e and profile page of a user (see Figure 4.3), there are four special buttons which are labelled as;

- **Buy**
- **Have**
- **Help**
- **Like**

Three of these buttons are the same we mentioned above on product profile page, and one additional "Help" button has been put there. This is also essential for TrendPin. The users can mention a product not only by just writing comments or attaching media to it, but also can specifically indicate whether they are going to buy, they already have, they request help, or they like that product. After creating the post, their friends can see it on their news feed screen and write comments to that post. This makes it easier to express the users thoughts to their friends, and much efficient. With this kind of sharing, friends of the users would like to leave comments related to specific reasons. For instance, if it is a **Help** post (see Figure 4.4), it is assumed that the user will get comments about helping to the specified problem.

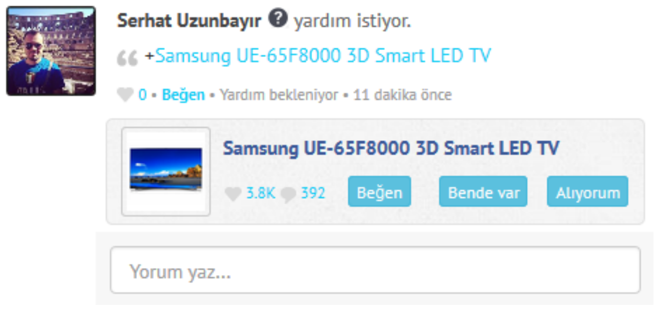

Serhat Uzunbayır yardım istiyor.
+Samsung UE-65F8000 3D Smart LED TV
0 • Beğen • Yardım bekleniyor • 11 dakika önce

Figure 4.4: A **Help** post about a product

4.2.2 Additional Features

There are additional features of TrendPin and they are listed below:

- What the users share can be also posted on their own Facebook profile page, if they enable it from the settings panel.
- TrendPin can suggest to the user to the products which they may like in their user profile page (see Figure 4.3) based on the previous activities of them as well as their friends on the application.
- It is also possible to compose and send instant messages to the friends by clicking to messages icon (see Figure 4.5), as well as reading the ones the

users get from their friends.

- The users get notified by any activity that happened about them and the products they interacted with, and observe these on the notification panel like other social networks such as Facebook and Twitter.

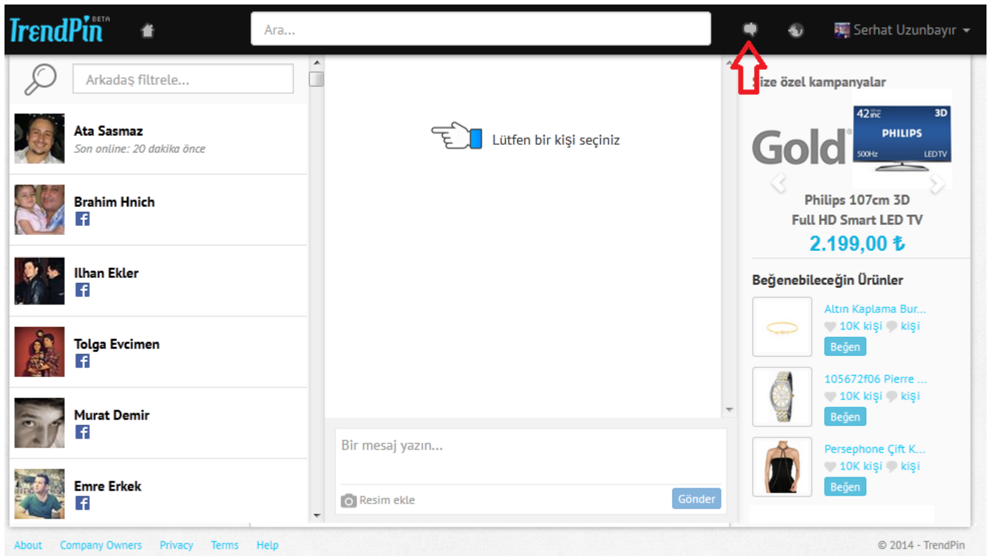


Figure 4.5: Messages Panel

TrendPin is a powerful online shopping platform with really different features than other platforms are providing today. Initial version of the system is designed with a relational database model. Since it is also a social networking platform with thousands of users and products, it is started to generate data large in volume. Users can do many activities using it at any time, so that velocity of creating data is high. Shared posts can vary by containing unstructured data as text or media files. Because of these reasons, TrendPin is a system that generates Big Data, thus making it very suitable to convert its database into a NoSQL database and do research for Big Data analysis.

4.3 TrendPin as a Big Data Project

When this project is started, TrendPin was just an idea. There were only discussions about the features, how to add more and improve them without considering design and implementation details. All we have were prototypes of screens, nothing more. As discussions went on, design and implementation details of the system emerged and we agreed on one important fact; *TrendPin is a Big Data Project*.

To be able to say that a system contains Big Data, there are couple of criteria to be considered. We need to look at the features of Big Data which we have discussed in detail in Chapter 2. What makes Big Data is the 3V's concept:

- As for *volume*, there are millions of datasets in the database of TrendPin that are comes from user interactions, products, and transactions. Hence, the volume is big.
- As for *variety*, many forms of data can be created, or uploaded to the system. These include text data during chatting via messaging interface and commenting on products, or other users' posts, uploading different types of media such as image and videos, inquiry data from searches, numerical data, status updates, and etc. Therefore, data generation is unstructured.
- As for *velocity*, users can post or interacted with anything at any time, product files are checking everyday regularly to update prices on various stores. The results should be available immediately. For these reasons, data creation speed is high.

In a usual online shopping application, there are products to sale and the users can interact with the system to choose, pay, and wait for their product to be delivered. Now, the definition of TrendPin includes social networking. Which means that users are not standing as individuals in the database. There is a friendship relation which creates connections among different users. As a result, this creates a platform to share common interests, ideas, requests, and more. When thousands, millions, even billions of people start to interact with such a system, there will be a tremendous amount of data generation over the cloud to be handled. As time passes, some portions of the data need to be transferred to other servers. Relational model fails to process such interactions and the system will surely get slower. That is the main reason of moving to NoSQL trends, and for this project, we wanted to try graph databases. We have already discussed that when a system contains a social networking procedures, graphs work the best.

4.4 Summary

In this chapter, we introduced TrendPin, which combines online shopping with social networking to provide an unusual shopping experience. We discussed core attributes of the system to differentiate it from other e-shopping applications. The system is social, meaning that the users can not only purchase items or services, but also be social by expressing their feelings, desires, opinions about their expenses, as well as getting knowledge of the products that their friends already have. Moreover, the users can message with their friends, get recommended of the products they can like and maybe buy in the future. We have also explained why TrendPin is a Big Data project. In the next chapter, we define different data models and analysis of the system.

Chapter 5

Different Database Models For TrendPin

5.1 Introduction

We have explained in Chapter 4 that there was not any implementation of TrendPin before we start this thesis work. It was challenging to start to develop such a system in terms of design, coding, and analysis due to the fact that everything was started from scratch. The aim of this thesis work is to design relational and graph models and test their performances on the system.

Initially, user interfaces of the system are created based on the screenshots. The main purpose of this phase was to create user-friendly and easy to use interfaces which provide all of the requirements within context of the problem. After user interfaces are agreed to work best with the system, implementation phase for the application began. Since the implementation is not the concern of this study, we are going to focus on database designs behind the system.

TrendPin is designed to work with a relational database management system in the first iteration. In this version, data is being held within relational tables. Second version of TrendPin is designed using graph databases. Data is being stored and handled using a graph structure. We explain how these two different data models are designed, compare query results, and discuss our findings throughout the rest of this chapter. Moreover, TrendPin was in under development throughout this thesis study. Therefore, it should be keeping in mind that TrendPin is supposed to keep improve by applying updates periodically. Therefore, in two of our versions, the system lacks some of the features such as registration with username and password.

5.2 Relational Data Model

In the first iteration, a relational model is designed using ASP.NET technology with Microsoft SQL Server database product. More importantly, before relational database model has been created, it has known that a graph model for this project is going to be designed. Therefore, this situation affected relational design decisions in terms of table separation. For instance, there is a table called `UserProductRelation` which corresponds to the relationship between graph nodes.

Relational model consists of fourteen different tables which are listed below. The reason behind this partition approach is to avoid violation of normalization rules which we have covered in Chapter 3. Thus, retrieving information sometimes requires many join operations and increases retrieval time inevitably. We explain table details by stating attributes, primary keys, relations, and data types covering all features from authentication to messaging in the following sections.

- User
- Product
- UserProductRelation
- Post
- PostLike

- FbFriendship
- ProductCategory
- CommentLike
- RelationType
- Comment
- FbProfile
- Follower
- Shop
- Vendor

The most important two entities of TrendPin are users and products. Posts that are created by users are another important entity category. Then comments on those posts comes to the agenda. There are supportive entities related to relation type between other entities, and suppliers.

5.2.1 Authentication

There is not an implementation of a system specific registration as we underlined before. Thus, users can only register via Facebook authentication module. If the user does not have a Facebook account, the system cannot be usable. Actually, this design decision was made based on announcing TrendPin to more people, especially to the friends of the users. Since TrendPin never went fully live, the number of users would be limited. We thought that if a user registers using Facebook accounts, more people will at least try the application. In addition, for such a young system, security is another important property. Facebook Login API is not only easy but also a reliable way of connection to an application.

When a user tries to connect with a Facebook account, the process returns a *Facebook profile id* which can be used to retrieve many information about that particular user. This id is a large number and can be stored as floats, or casting to string. However, the best option to store them as BIGINT(64). It is enough for TrendPin to store ids. Therefore, **FbProfile** table (see Figure 5.1) stores only Facebook ids in `FacebookProfileId` as BIGINT(64), names of the users in `FullName` as strings, and profile photo URLs in `PhotoUrl` as strings.

In addition, **FbFriendship** table (see Figure 5.1) keeps Facebook friend lists of users to suggest invitations of TrendPin if they have not registered yet. This table has an `Id` to increments friends as an integer, `FbProfileId` of the user, and a user id given by the system in `UserId` as integers.

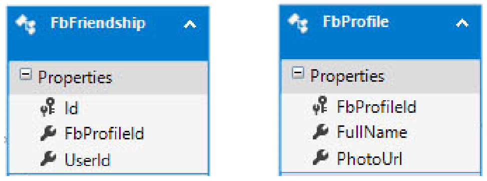


Figure 5.1: `FbFriendship` and `FbProfile` tables

5.2.2 Users and Products

Users and products are vital entities of TrendPin. Their attributes, and relationships include many details and causes lots of transactions during system usage.

A user is an essential entity of any system. On TrendPin's database details of them is stored in **User** table (see Figure 5.2). It holds user ids in`Id` as integers, first name of the users in `First Name` as strings, last name of the users in `Last Name` as strings, email addresses in `Email` as strings, login keys to be used for login processes after registering with Facebook ids in `LoginKey` as strings, photos of users in `PhotoFile` as strings, last login time of users in `LastLogin` as DateTime format, Facebook user names in `FbUsername` as strings, and Facebook profile ids in `FbProfileId` as BIGINT(64).

A product is another essential entity of an online shopping platform. On TrendPin's database details of them is stored in **Product** table (see Figure 5.2). It stores product ids in `Id` as integers, name of the products in `Title` as strings, details belong to products in `Description`, product seller ids in `VendorId` as integers, whether products has images or not in `HasPhoto` as boolean values, photo URL of products in `RemotePhotoURL` as strings, product category ids in `CategoryId` as integers, prices of products in `Price` as integers, shop ids in `ShopId` as integers, URLs of the products in the website of the shops which are needed to be redirected to those pages in `ShopProductUrl` as strings, counting likes on the products in `LikeCount` as integers, counting owners of the products in `OwnCount` as integers, counting the number of users who wants to have these products in `WantCount` as integers, counting help requests of the products in `HelpCount` as integers, to keep insertion time of the products in `InsertDate` as DateTime format, and the last update time of the products in `UpdateDate` as DateTime format.

A relationship between a user and product can be different. A user can like, buy, have, or need help about a product. For this reason, **UserProductRelation** table (see Figure 5.2) keeps ids to count relations in `Id` as integers, user ids in `UserId` as integers, product ids in `ProductId` as integers, relation ids in `RelationId` as integers, and the time these relations are created in `CreationDate` as DateTime format.

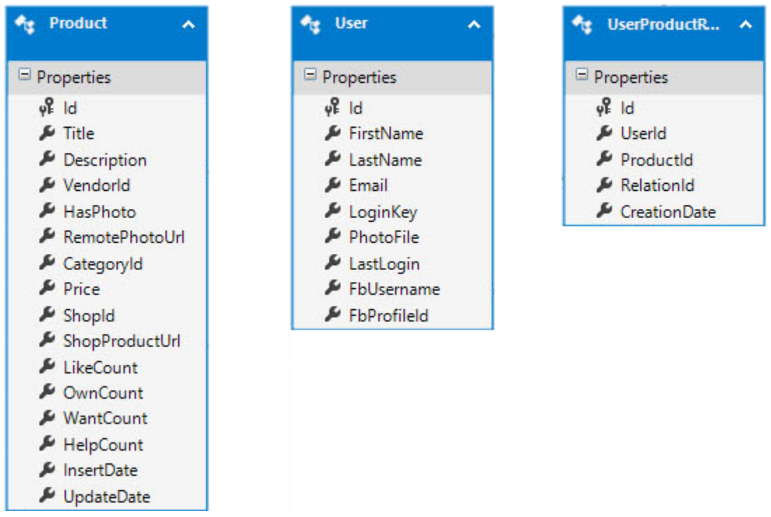


Figure 5.2: `Product`, `User`, and `UserProductRelation` tables

5.2.3 Posts and Comments

Posts and comments are important features of communication and sharing thoughts among users. In terms of social networking, they do not refer to the same object. A post is always created first by a user, and others create comments below this posts. That's why they are distinguished from each other and stored in different tables.

A post can be created by users about products and stored in **Post** table (see Figure 5.3). It stores ids of the posts in `Id` as integers, ids of the users who created the post in `UserId` as integers, product ids which are related to the post `ProductId` as integers, contents of the post if they do not include any media in `Text` as strings, media included content of the posts in `FormattedText` as strings, relation ids in `RelationId` as integers, deciding whether these posts are help posts or not in `IsHelpPost` as boolean values, counting likes of the posts in `LikeCount` as integers, counting comments left under the posts in `CommentCount` as integers, and the creation time of the posts in `Date` as DateTime format.

A comment can be created by users under posts as answers and stored in **Commet** table (see Figure 5.3). This table stores ids of the comments in `Id` as integers, ids of the users in `UserId` as integers, content of the comments in `Text` as strings, counting likes of the comments in `LikeCount` as integers, and the creation time of the comments in `Date` as DateTime format.

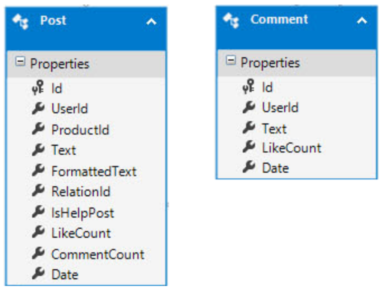


Figure 5.3: `Post` and `Comment` tables

5.2.4 Shops and Vendors

Shops are the places which sell different kinds of products, whereas vendors are small individual suppliers to provide goods or services. They are separated from each other to ensure normalizations are under control.

A shop information is given by the administrators and kept in **Shop** table (see Figure 5.4). It stores ids of the shops in `Id` as integers, name of the shops in `Name` as strings, and URL's of the shops in `Website` as strings.

A vendor information is also given by the administrators and kept in **Vendor** table (see Figure 5.4). It stores ids of the vendors in `Id` as integers, name of the vendors in `Name` as strings, and URL's of the vendors in `Website` as strings.

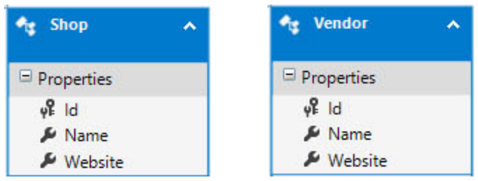


Figure 5.4: `Shop` and `Vendor` tables

5.2.5 PostLike and CommentLike

A post and a comment can be liked by users and this should be stored in case there is a need when finding who liked which post and comment (see Figure 5.5).

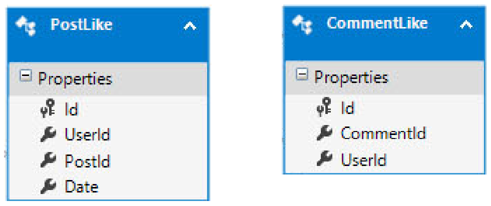


Figure 5.5: `PostLike` and `CommentLike` tables

5.2.6 Product Categories and Followers

There are different kind of products in the database such as notebooks, mobile phones, accessories, and etc. This information could be kept in `Product` table too. However, it was necessary to separate these two entities to help listing products as a tree structure. For example; *products* → *technology* → *electronic* → *music* → *albums*.

Categories of a products are kept in `ProductCategory` table (see Figure 5.6). It includes id of the product in `Id` as integers, name of the products in `Name`, and up level category in `ParentId` as integers.

Since TrendPin has four distinct post types (buy, have, help, like), when a post has been created, type of the post should be kept within this entity.

In social network applications, a user may follow another user's profile page to observe or get notified by actions. Here, we use this table to model friendship relations among users. `Follower` table (see Figure 5.6) holds this information. It contains ids of users in `UserId`, ids of the followers in `FollowerId`, and the time that they followed in `AssociationDate` as DateTime format.

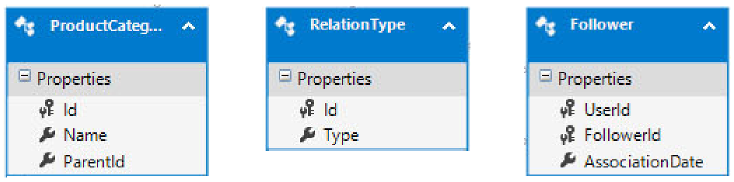


Figure 5.6: `ProductCategory`, `RelationType`, and `Follower` tables

5.3 Graph Data Model

In the second iteration, a graph model designed with Neo4j. Integration of Neo4j with ASP.NET provided by Neo4jClient driver. It was designed after relational model, therefore knowledge of the application domain was good. However, graph design was started from scratch which means from board designs to implementation road was long. Luckily, designing graphs is easier than relational tables, since they are not only just an abstraction of real life entities with nodes and relationships, but also visual transferring is possible.

In our graph data model, there are users and products as nodes, and social relationships among them. TrendPin has different features varying from commenting on products to messaging between users. Therefore, we divided our data model into six parts;

- Relationships between users and products
- Friendship among users
- Relationship between products and shops
- Creating posts and comments
- Relationship between products and campaigns
- Private messaging

5.3.1 Relationships Between Users and Products

Users and products are different two main entities. Their key-value properties are similar to the attributes of relational model.

User (**GraphUser**) information is generated during Facebook authentication process. A user has many attributes and can be connected to many other entities. User objects include ids in `Id` as integers, full name of the users in `FullName` as strings, name of the users in `FirstName` as strings, surname of the users in `LastName` as strings, to indicate whether the users are members or not in `IsMember` as booleans, to indicate whether the users are administrators or not in `IsAdmin` as booleans, email addresses of the users in `Email` as strings, keys which are necessary to login are in `LoginKey` as strings, profile photo URLs in

`PhotoFile` as strings, latest login time in `LastLogin` as Date, Facebook user names in `FbUsername` as strings, to indicate whether the users are promoted to not in `IsPromotedUser` as booleans, register time in `RegistrationDate` as Date, when the last message received in `LastMsgReceivedOn` as Date, when the friend lists are updated in `LastFriendsUpdate` as Date, tokens related to last access time in `LastAccessToken` as strings, time of last access tokens in `LastAccessTokenIntTime` as Date, time when the last activities have occurred in `LastActivityEpochTime` as Date, and to indicate the number of unread notifications in `UnreadNotificationCount` as integers.

A Facebook login also returns friend lists of the users. With that friend list, graph model creates `[IS_FRIEND_OF]` relationships between users after they login to TrendPin for the first time. When a friend is already registered to the database, `[IS_FRIEND_OF]` relationship is automatically created on the graph model.

Product **GraphProduct** information comes from vendors and shops as XML files. After parsing them, GraphProduct nodes are created. These objects include id of the products in `Id` as integers, id of the shops and products together in `ShopProductId` as string, name of the products in `Title` as strings, description of the products in `Description` as strings, ids of the shops only in `ShopId` as integers, ids of the vendors only in `VendorId` as integers, to indicate whether products have images or not in `HasPhoto` as booleans, URLs of the images in `RemoteRphotoUrl` as strings, category ids to represent which category these products belong to in `CategoryId` as integers, price of the products in `price` as integers, URLs of the shops where these products are sold in order to redirect when clicked in `ShopProductUrl` as strings, the number of likes in `LikeCount` as integers, the total number of have selections in `HaveCount`, the number of buying selections in `BuyingCount` as integers, the number of need help selections in `NeedsHelpCount` as integers, the time when these products inserted to the database in `InsertDate` as Date, the time when these products are last updated in `UpdateDate` as Date, and to indicate whether products are deleted for certain reasons in `IsDeleted` as boolean formats.

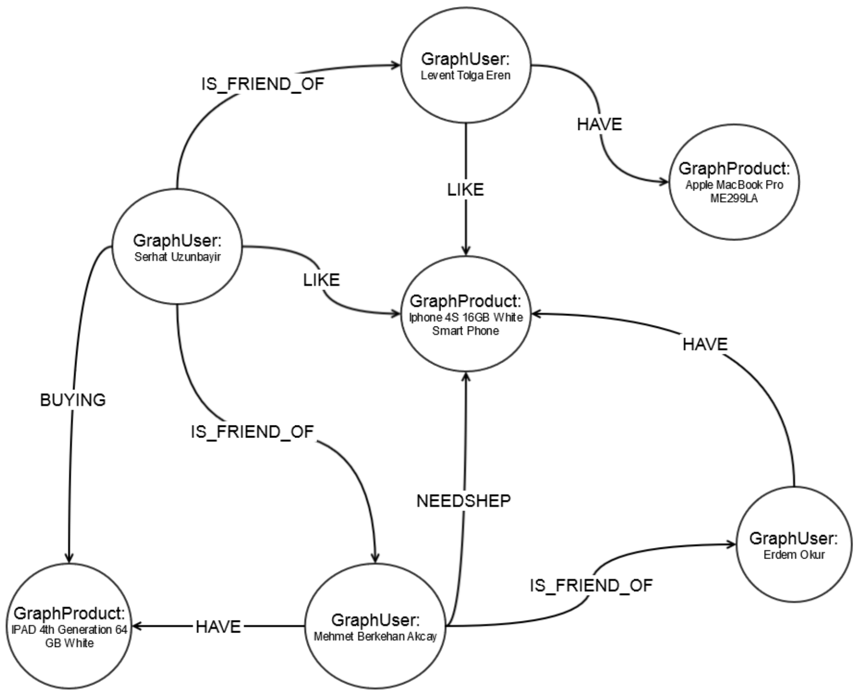

GraphUser: Levent Tolga Eren
GraphProduct: Apple MacBook Pro ME299LA
GraphUser: Serhat Uzunbayir
GraphProduct: Iphone 4S 16GB White Smart Phone
GraphUser: Erdem Okur
GraphProduct: IPAD 4th Generation 64 GB White
GraphUser: Mehmet Berkehan Akcay
IS_FRIEND_OF
HAVE
LIKE
LIKE
HAVE
BUYING
IS_FRIEND_OF
NEEDSHEP
IS_FRIEND_OF
HAVE

Figure 5.7: Relationship between users and products

A Relationship between a product and a user can be `BUYING`, `HAVE`, `LIKE`,or `NEEDSHELP`. When users select one or more of these options, relationships are created according to chosen types. Here is an example scenario;

"Serhat is a friend of Levent Tolga and Mehmet Berkehan. He likes Iphone 4S and is buying IPAD 4th generation. Levet Tolga has an Apple MacBook Pro and likes Iphone 4S. Mehmet Berkehan is a friend of Serhat and Erdem. He needs help with Iphone 4S and has IPAD 4th generation. Erdem has an Iphone 4S."

Figure 5.7 shows nodes and relationships among users and products in TrendPin according to the given scenario.

5.3.2 Friendship Among Users

Graphs are very good at designing friendship relationships without any confusion. Both storing and retrieving such kind of information from graphs are easy. Users can become friends with each other on TrendPin. We already show that the relationship is called `[IS_FRIEND_OF]`.

Let us have a closer look at friendships considering the following scenario;

"Sercan and Serhat are friends of Mehmet Berkehan. Serhat is also a friend of Levent Tolga. Moreover, Mehmet Berkehan is a friend of Erdem."

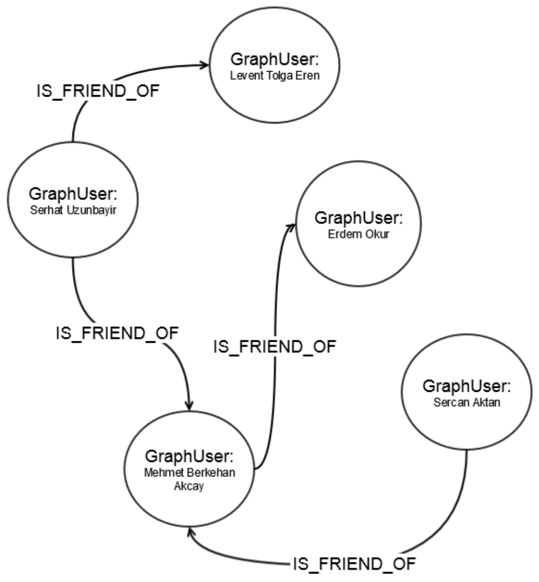


Figure 5.8: Friendship among users

Here, friendship seems to be in one direction. However, it is allowed to retrieve data from both sides of the relationships in Neo4j. Therefore, there is no need to create these relationships with both directions.

5.3.3 Relationship Between Products and Shops

Products are related with users, and also they can be sold in shops. We have already underlined that product pages include direct links to the shops which users can click and be redirected to the pages of selected shops. TrendPin uses XML files to gather product information from more than 20 shops at the same time including companies such as Gold, Dekoreko, Altinci Cadde, and Teknosa.

Shops (**GraphShop**) contain ids of the shops in `Id` as integers, names of the shops in `Name`, as strings, websites of the shops in `Website` as strings, URLs of XML files of the shops in `XmlUrl` as strings, type of the parsers in `ParserTypeName` as strings, total number of the products in `ProductCount` as integers, and time when the last update has occurred in `LastXmlUpdateTime` as Date formats.

In order to design products and their shops with graphs requires `[HAS]` relationships. Let us illustrate the following scenario on the graph 5.9;

"Gold and Teknosa sell Vestel V-Press Ütü, and Teknosa also sells Samsung LCD TV together with Darty."

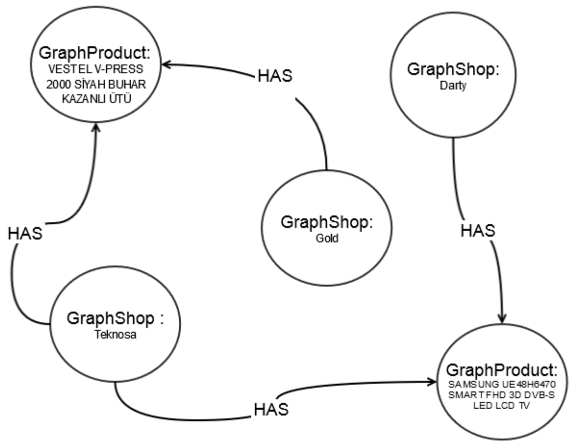


Figure 5.9: Relationship between products and shops

5.3.4 Creating Posts and Comments

Perhaps the most complex part of graph model of TrendPin is to have a solid, and consistent graph for creating posts and comments related to that posts.

A user can create a post about a product by selecting one type of buy, like, have, or help features. When a post is created, it is added to the database as a **GraphPost** labelled node which also includes type of the post as an attribute (buy, like, have, or help). Doing such a setting provides that a post will be stay at the top of all relevant comments. Thus, it will indicate its meaning clearly. In addition, post nodes have post ids in `Id` as integers, Facebook ids of the users in `FacebookId` as integers, id of the users who create those posts in `userId` as integers, products that are relevant to those posts in `ProductId` as integers, message contents in `Text` as strings, media of the posts if they included in `FormattedText` as strings, creation date of the post in `Date` as integers.

A user can create a comment under a post. Difference of a comment from a post is that comments do not have any types. Just exclude type property from a post and it becomes a comment in the database. Comment (**GraphComment**) nodes have ids in `Id` as integers, products that are relevant to those posts in `ProductId` as integers, ids of the users who create those comments in `UserId`, message contents in `Text` as strings, to count total likes of the posts in `LikeCount` as integers, and creation date of the post in `Date` as Date formats.

Relationships are another important design decision here. When a user posts a post, `POSTS` relationship is created between a user and a post. Since this post will be about a product, it is connected `TO` a product or more products. When a user creates a comment to a post, `WRITES` relationship is created between a user and a comment. Since this comment is written under a post, it connects to that post with `REPLIED_TO` relationship. If there are already comments above a new comment, `PREVIOUS_TO` relationship is created to indicate there are more comments before this one. Moreover, a post can be `LIKED` by a user.

These relationships are ternary relationships. The order is as follows:

- A user posts a GraphPost to a GraphProduct
- A user writes a GraphComment replied to a GraphPost
- A user writes a GraphComment replied to a GraphPost but also previous to a GraphComment

The scenario below projected on Figure 5.10 clarifies everything we discussed about posts and comments.

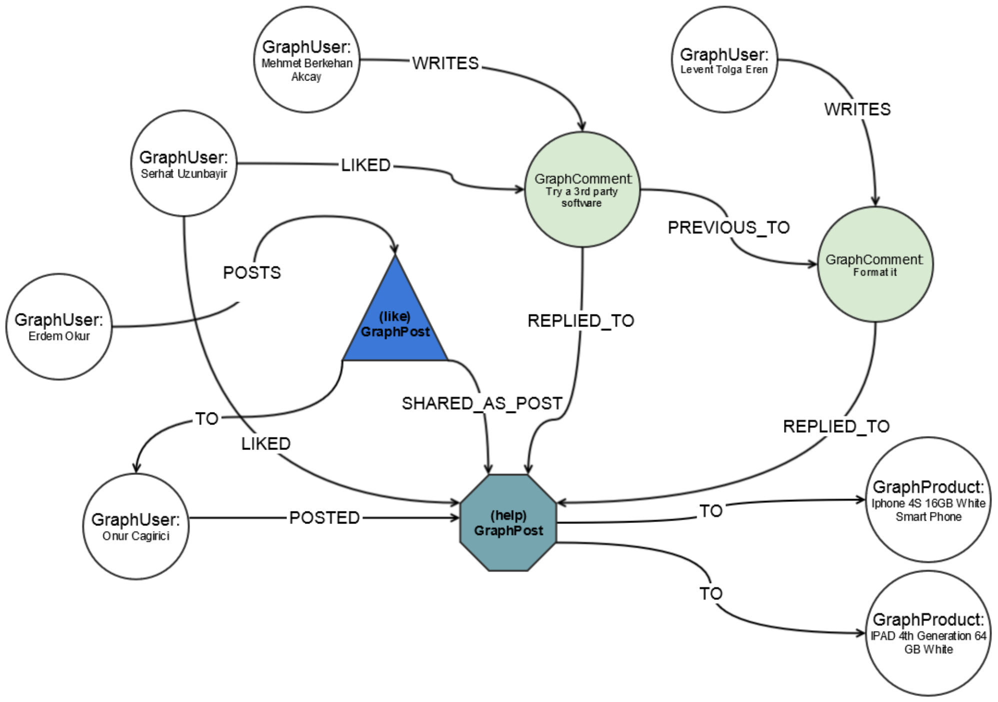

GraphUser: Mehmet Berkehan Akcay
WRITES
GraphUser: Levent Tolga Eren
WRITES
GraphUser: Serhat Uzunbayir
LIKED
GraphComment: Try a 3rd party software
PREVIOUS_TO
GraphComment: Format it
POSTS
GraphUser: Erdem Okur
(like) GraphPost
REPLIED_TO
SHARED_AS_POST
REPLIED_TO
TO
LIKED
GraphUser: Onur Cagirici
POSTED
(help) GraphPost
TO
GraphProduct: Iphone 4S 16GB White Smart Phone
TO
GraphProduct: IPAD 4th Generation 64 GB White

Figure 5.10: Creating posts and comments

"Onur creates a post of help about a Iphone 4S and IPAD 4th Generation. Levent Tolga writes a comment to that post as a reply. Mehmet Berkehan writes a comment after Levent Tolga's comment, but it is still a reply to Onur's post. Serhat may like Mehmet Berkehan's comment and Onur's post. Erdem posts a post of like as a reply of Onur's post but it can be shareable as a post."

It seems quite complex on the example, however it is quite reasonable to link all of users, posts, comments with that kind of structure. A query to fetch data can be written easily with to traverse this graph to find what is needed.

5.3.5 Relationship Between Products and Campaigns

TrendPin started to suggest products to its users when graph model has been created. To provide this feature, campaign (**GraphCampaign**) objects are created. This object includes products which can be shown to users who have already searched or purchased similar products. Campaign objects contain ids of the campaigns in `Id` as integers, product ids in these campaigns in `ProductId` as integers, finish time of campaigns in `EndDate` as Date, and product objects within campaigns in `Product` as GraphProduct formats.

`FOR` relationship is created when a product needs to be contained within a campaign.

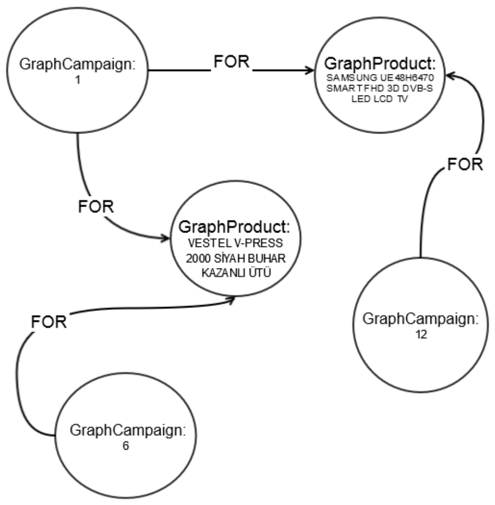


Figure 5.11: Relationship between products and campaigns

The scenario below and Figure 5.11 illustrates how this feature is modelled.

"Campagin 1 and 12 contain Samsung TV, campaign 1 and 6 contain Vestel V-Press Ütü."

5.3.6 Private Messaging

Private messaging another feature of social networking. It is needed to track how private messages are being sent and stored within entities. First, a user prepares a message to another user and start a conversation by sending that message. After a conversation is started, they can continue sending messages to each other without waiting an answer similar to texting via smart phones' messaging feature. For example one can send or receive 4 successive messages before getting a single reply.

Message (**GraphMessage**) nodes have id of the message in `Id` as integers, ids of users who first create those messages `FromUserId` as integers, ids of the users who receive these messages in `ToUserId` as integers, contents of the messages in `Text` as strings, product ids which these messages are about in `ProductId` as integers, creation date of the messages in `Time` as Date, and IPs of te users in `SentIP` as strings.

Relationships are similar to posts and comments, and the order is as follows:

- User X `WRITES` a message K to user Y
- User Y writes message L `REPLIED_TO` message K
- User X writes message M replied to message K but `PREVIOUS_TO` message L

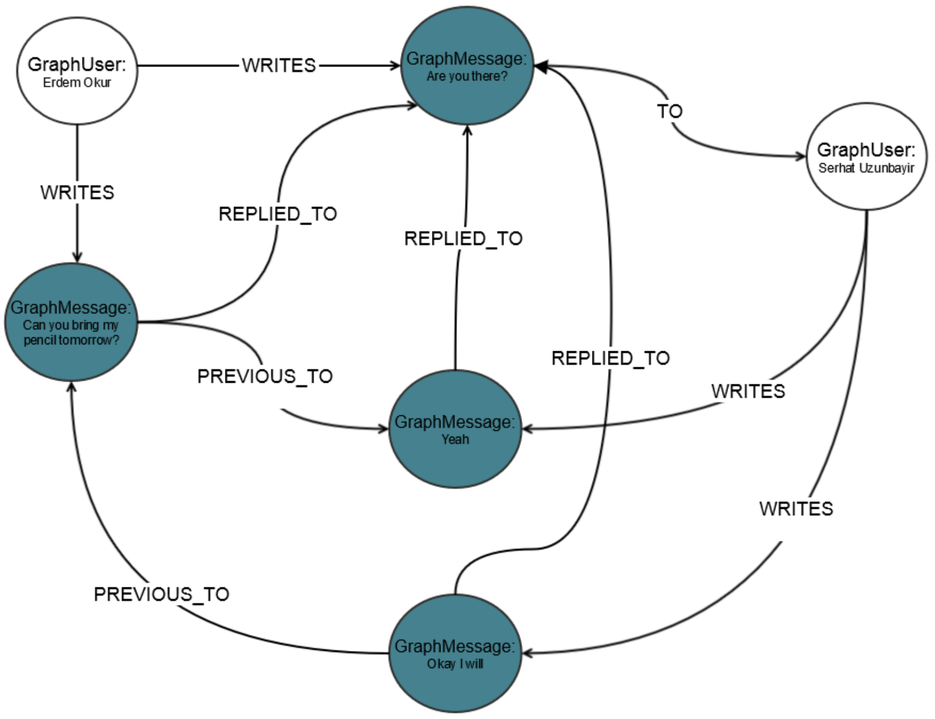


Figure 5.12: Private messaging

Story below is an example of a private conversation. Figure 5.12 illustrates how private messaging is modelled as a graph.

"Erdem writes a message to Serhat for asking whether he is online or not. Serhat replies back and Erdem asks another question and conversation goes on."

The graph indicates who sent which messages to who, also points to previous messages with ternary relations. When it comes to querying and displaying these messages, this structure will make it simpler.

5.3.7 Special Cases of Graph Model

After graph model is completed, we wanted to move on with testing. Second iteration of database design seem fine at first, however the graph design was not felt complete. It was not answering the following crucial questions:

- What about realizing same products from different shops and vendors? How can it prevented to create new nodes for the same items?
- What about categorizing products of products such as accessories?

To be able to provide a solid graph model, we had to modify it by eliminating such problems. Therefore, we introduce *information extraction* concept in Chapter 6. With the additions of this concept, graph model will be completed.

5.4 Summary

In this chapter, we discussed two different data model designs and implementations of TrendPin. We showed tables and their attributes in relational data model, entities and relationships among them in graph model. We also explained what each attribute stands for, as well as provided examples to increase understanding. We agreed that there is not a single design that fits for all requirements of any system.

Relational models are advantageous because they are in the market for more than forty years and they have certain steady rules. On the other hand, graph databases are much newer in the area. They work better on Big Data problems for sure. We compare performances in Chapter 7, however before doing that, we discuss information extraction concept in the next chapter to overcome problems of graph model and complete it.

Chapter 6

Information Extraction

Information systems are being filled with huge amount of textual data which are coming from multimedia communications. With the increase of unstructured data that is being generated by users throughout blogs, social platforms, and other websites, many of those systems are not able to extract useful information for their specific purposes. The extraction of useful data and convert that into meaningful entities within the system requires specific techniques. “Developing intelligent tools and methods, which give access to document content and extract relevant information, is more than ever a key issue for knowledge and information management.” (Nédellec C. and Nazarenko A., 2006)[78] To be able to satisfy such a requirement, information extraction (IE) concept emerges as one of the most important areas to be researched.

6.1 Definition

Information extraction is a very crucial concept when searching, indexing, and matching patterns through a huge amount of data. Origins of the problem comes from generation of relational data by using natural language texts in 1970s (Wu F. and Weld D.S., 2013)[86]. The main research areas of information extraction is natural language processing, web mining, and information retrieval. Furthermore, there are huge domain-specific applications which IE is an essential work to be done such as biological literature mining, and financial analysis.

6.2 Method Explanation

Basically IE systems are focused on searching information that is related with what the user needs to see.[66] The aim is to identify the names, objects, and their relations between those entities from unstructured texts. "Two fundamental tasks of information extraction are named entity recognition and relation extraction." (Jiang, J., 2012) [68]. Considering we have a sentence as "Last week, Microsoft corporation has announced that they have agreed to establish a partnership with Pargesoft and COMEL.". Now, there will be a function called *information extractor* which contains some arguments to perform a *named entity recognition* and relations between these to satisfy a *relation extraction.* The goal of this extractor is to generate two arguments and a relation as a triple for all other relations. This summary information will then help to solve problems regarding to the issue. Here we would have such a function;

partnership(company1, company2, date)

It can be said that IE is a text exploration method (Nédellec C. and Nazarenko A., 2006)[78]. Another example in Figure 6.1 explains quite well about how information extraction tidies up unstructured data.

Voorhees (1999) explains that there are two main activities in information retrieval systems; indexing and matching.

- *Indexing* is the process of finding some of the suitable entities to represent texts.
- *Matching* is the process of finding the similarities between two distinct representations.

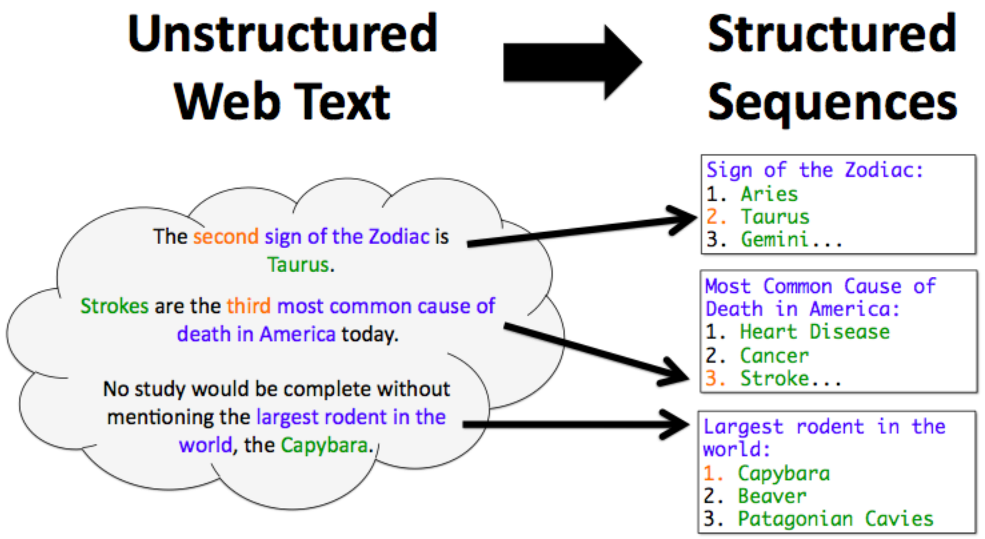

Unstructured Web Text
Structured Sequences
The second sign of the Zodiac is Taurus.
Strokes are the third most common cause of death in America today.
No study would be complete without mentioning the largest rodent in the world, the Capybara.
Sign of the Zodiac:
1. Aries
2. Taurus
3. Gemini...
Most Common Cause of Death in America:
1. Heart Disease
2. Cancer
3. Stroke...
Largest rodent in the world:
1. Capybara
2. Beaver
3. Patagonian Cavies

Figure 6.1: Information extraction example[21]

When these two operations are done and analysed successfully, with the help of additional structures such as knowledge bases and etc., systems can work smarter and more efficient. Based on this assumption, we have implemented an IE system for TrendPin, and explain the details in the following sections.

6.3 Issues with Graph Model of TrendPin

Big Data contains semi or unstructured information as we discussed in Chapter 2. That information is meaningless, unless it has been processed with the help of the methods we explained in the previous section. Now, we have a huge data collection supplied by different vendors within our graph database. We wish to somehow extract structured information from that data in order to direct our system operating smarter. By doing so, our goal is to improve searching throughout the graph database and display the results to the users including meaningful information *automatically*.

While we were analysing our graph database design we faced with three significant problems:

- Automatic links between related products
- Similar products from rival vendors
- Displaying smarter results

6.3.1 Issue I: Automatic Links Between Related Products

In graph database of TrendPin's, we have many different products from mobile phones to televisions and even clothes. We get our data from different vendors in XML format. After we parse the file and insert into the graph database, we create relations among different users, and also between products and users themselves. Up to this point everything seemed appropriate for our purposes. However, when we think about how an accessory is related to its product, the design has stuck. In addition, when we want to divide items into distinct categories, same situation would seem continue to occur.

Figure 6.2: iPhone and some of its accessories

For instance, an iPhone 5S case is an accessory of an iPhone 5S. It cannot be used on other iPhones. At the same time, an iPhone 5S charger and an iPhone headphones is also accessories of iPhone. There are many more items which should be treated as accessories of some other products such as tripods, selfie sticks, cases, screen protectors, and etc. Additionally there are some exceptional cases that should be taken into account; an iPhone 4S charger cannot be used on iPhone 5S. Headphones can also be an accessory of MP3 players, computers, and tablets. *How can we decide which item is an accessory of another item automatically?*

6.3.2 Issue II: Similar Products From Rival Vendors

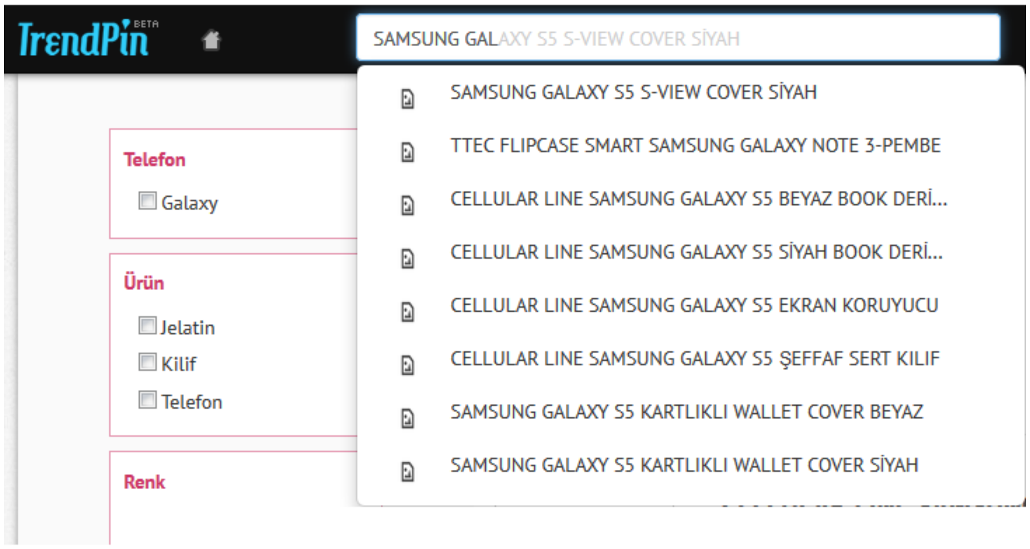
We said that we get our data in XML format. After parsing these files, the system updates relevant nodes and relationships if they are already exist in graph database. If they don't exist, it creates new ones. Since we don't have only one vendor or supplier, we get XML files written in distinct formats. For example one tuple of a product from a vendor includes it's web link, price, image, model, and full name respectively. Whereas, another tuple for the same product from another vendor includes product's id, web link, price, category, image, full name, model, description, and credit card purchase choices. Therefore, this data cannot be matched exactly when a complete keyword matching operation has been performed due to the difference of order and attributes of tuples. Moreover, full name of the same product can also be different in different vendors. Below is an example of product names in three different XML files;

- iPad Mini 64 GB Wi-fi White MD533
- iPad Mini 64GB Wi-Fi White Tablet PC
- iPad Mini 64GB White

In the first one, full name of a product is written with a model number MD553 which we have no idea about what it means. In another XML, the same product is labelled with "Tablet PC" indicator. In the last one, it is not stated that an iPad Mini has Wi-Fi (we know that iPad Mini has Wi-Fi feature by default). Although all of these three products are representing the same product, the strings are not completely identical. Therefore, exact matching is not possible. *How can we be sure about these three products are actually the same?*

6.3.3 Issue III: Displaying Smarter Results

TrendPin has a very huge database with thousands of different types products. When users search a product, the system should be able to display the best and relevant results for the users at the top. For example, one may want to find information about a smart phone or its accessories such as Samsung Galaxy S5 and start to type "SAMSUNG GAL", the system should try to auto complete it.



SAMSUNG GALAXY S5 S-VIEW COVER SİYAH

SAMSUNG GALAXY S5 S-VIEW COVER SİYAH
TTEC FLIPCASE SMART SAMSUNG GALAXY NOTE 3-PEMBE
CELLULAR LINE SAMSUNG GALAXY S5 BEYAZ BOOK DERİ...
CELLULAR LINE SAMSUNG GALAXY S5 SİYAH BOOK DERİ...
CELLULAR LINE SAMSUNG GALAXY S5 EKRAN KORUYUCU
CELLULAR LINE SAMSUNG GALAXY S5 ŞEFFAF SERT KILIF
SAMSUNG GALAXY S5 KARTLIKLI WALLET COVER BEYAZ
SAMSUNG GALAXY S5 KARTLIKLI WALLET COVER SİYAH

Figure 6.3: Searching a product using TrendPin

Auto completing is not a challenge here because it is easy to implement. However, when there are many variations of the same product such as white, black, blue, and garnet red in color or 16 GB, 32 GB, and 64 GB in memory, an another issue emerges. Different users care differently about those variations. Especially for Apple products, size of the memory is an affecting factor for users when buying, since it cannot be changed later on. *How can we decide which one should appear at first?*

6.3.4 Solution

When we combine and think about those three independent issues, we can see that there is a common feature; all of them are about products, not users; specifically they are about *"product names"*. And the product information comes from XML files as *texts*. These texts are actually unstructured data and if we can bring them a meaning, our issues will then be solved naturally. The situation here leads us to the information retrieval techniques which we are explaining since the beginning of this chapter. There is a term called *"knowledge base"* which will be terrifically helpful to solve our issues.

Knowledge bases and information retrieval techniques indeed helps to fasten searching processes. On the other hand, using an intelligent subsystem, search results are now much more better for any users of the system. They help to analyse the problems that are arising from information extraction. Another importance of a knowledge base is that it stays as a huge information resource for the future learning to make data more meaningful. Our primary goal is to understand what a knowledge base is, and then build one specifically for our system. After that, we will perform *"knowledge based information retrieval methods"* to overcome the problems we listed before.

6.4 Knowledge Bases

A knowledge base is a repository which is organized in a certain form depending on information retrieval and may be based on both a human controlled process which consists of textual physical documents, or an artificial intelligence controlled process which operates automatically.

"A knowledge base typically contains set of concepts, instances, and relations" (Deshpande et. al, 2013)[52]. In the example below, there is a set of concepts including all, schools, countries, elementary, secondary, and cities. These concepts arranged in a hierarchical order starting from the most general to the most specialized words. Instances are known examples of the most specialized concepts such as Hillcrest Public School (it is an elementary school in London), Saunders Secondary School (it is a secondary school in London), and London (it is a city). There are also relationships between them to indicate the type of instantiations of instances from concepts.

Generally there are two kinds of knowledge bases; domain-specific knowledge bases and global knowledge bases. Deshpande et. al. (2013)[52] underlines

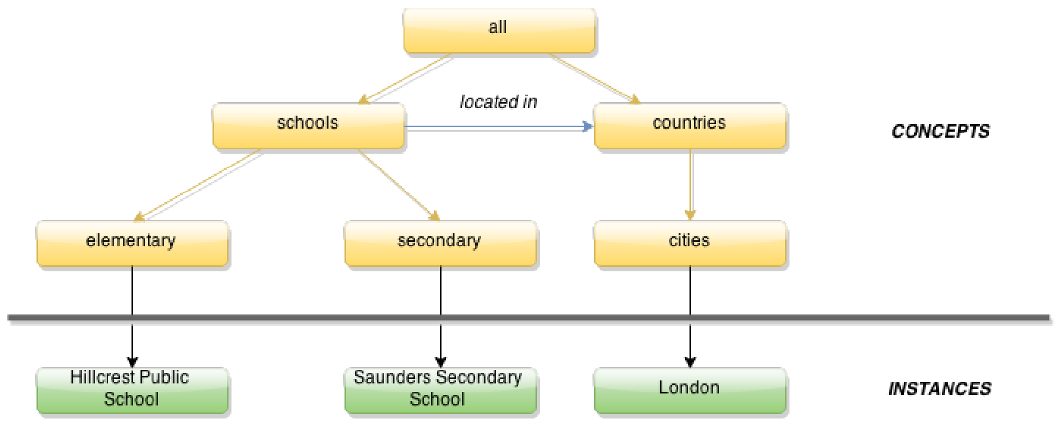


Figure 6.4: An example of a Knowledge Base

that Google Scholar and especially product based databases which are built by e-commerce web systems are domain-specific knowledge base examples. In our case we have to build and use a domain-specific knowledge base because of TrendPin's current structure.

Other examples of knowledge bases can be listed as follows;

- http://whatis.com/
- http://docs.nexcess.net/
- http://www.time4advice.co.uk/kb/
- http://knowledgebase.mediafire.com/
- http://helpdesk.nex-tech.com/
- http://www.ctera.com/kb/
- http://realestateexpress.com/
- http://www.efsumb-portal.org/ep/

These websites contain huge amount of meaningful data and are still open to research for the same purposes.

6.5 A Knowledge-Based Information Retrieval Module for TrendPin

There are different approaches when creating a knowledge base for an application. Each approach requires distinct steps to be completed. However, all of those will fulfill the same goal. For this thesis work, we generalized entire knowledge base creation process within three distinct steps:

- *Step 1: Creating a domain-knowledge*
- *Step 2: Automatic feature extraction*
- *Step 3: Combining knowledge base with keyword search*

6.5.1 Creating a Domain-Knowledge

Domain-knowledge is basically a text file which contains concepts and their associated values. Before creating a domain-knowledge, we ran some queries over our database and found word frequencies of the products which we have. This information helped us to find more reliable entities and bought us some time.

Word	*Frequency*
kol	13202
saati	13177
erkek	8579
kolye	5677
bayan	5660
siyah	5532
ml	4939
kadın	4458
bileklik	4148

Table 6.1: Top ten word frequencies

After fetching the most used words, we classified and formed two groups as concepts and values for our knowledge domain. At the same time, we decided that some of the categories should be major ones whereas some of them should be minor ones. Because this will be very useful to solve our third issue *(displaying smarter results)*. With the help of word frequency list, we built the domain-knowledge on Table 6.2.

Category	*Value*
color	beyaz, kırmızı, siyah, pembe, bordo, siyah, mavi, mor, turuncu, yeşil, gri, mor, kahverengi, turkuaz, sarı, lacivert, blue, white, black, gold, silver, rose, şeffaf
gender	erkek, kadın, unisex, men, women
usingPlace	kol, duvar
appearance	taşlı, desenli, detaylı, baskılı, çizgili, çiçekli, leopar, patchwork, dantel, pırlantalı
material	gümüş, altın, kaplama, deri
measure	m, cm, mm
supplier	casio, asus, apple, samsung, nokia, hp, armani, nacar, nikon, sony, bosch, philips, siemens, arçelik, lacoste, vestel, esprint, toshiba
size	mini, midi, büyük, küçük, uzun, kısa, mega, micro, macro
model	galaxy, iphone, ipad, tablet, notebook, xperia, laptop, pc, bilgisayar
weightML	ml, 100ml, 50ml, 30ml
clothing	gömlek, pantolon, body, t-shirt, sweatshirt, tunik, kazak, bluz, eşarp, triko, ayakkabı, bot
weight	gr, kg
memory	8GB, 16GB, 32GB, 64GB, 128GB, 512GB, 1TB, 2TB
shaper	kare, dıkdörtgen, üçgen, daire
phone	smart phone, regular
size	ön, arka, önü, arkası

Table 6.2: A part of TrendPin's domain-knowledge

6.5.2 Automatic feature extraction

Now that we have a kind of a digital library as a domain-knowledge, we can start to do information extraction processes. Manolescu (1998) [75] explains that to be able to handle with complex data, large information, and perform similarity searching systems are forced to *understand* complex information and enable using it efficiently including faster response times. By doing that, we are be able to solve our problems in the graph structure, and push its abilities to the next level.

In our original graph database design, we had users and products as nodes, and relationships between users and products as well. To be able to do an information extraction from the knowledge base we have just created, we create links among products themselves. To do so, we use concepts and values as attributes for products, and then ask questions about how similar two products are. Let us explain how automatic information extraction is being done by the system;

Assume that we have two products labelled as X and Y. Normally these two products have their own attributes within the nodes. Right now, we are separating some attributes from those products and make them new standalone nodes. Labels of these nodes comes from categories. In this case, we have three categories which determine product X and Y in color, supplier, and memory. The connections between attribute nodes and products are has_color, has_supplier, and has_memory respectively. Figure 6.5 illustrates whole structure.

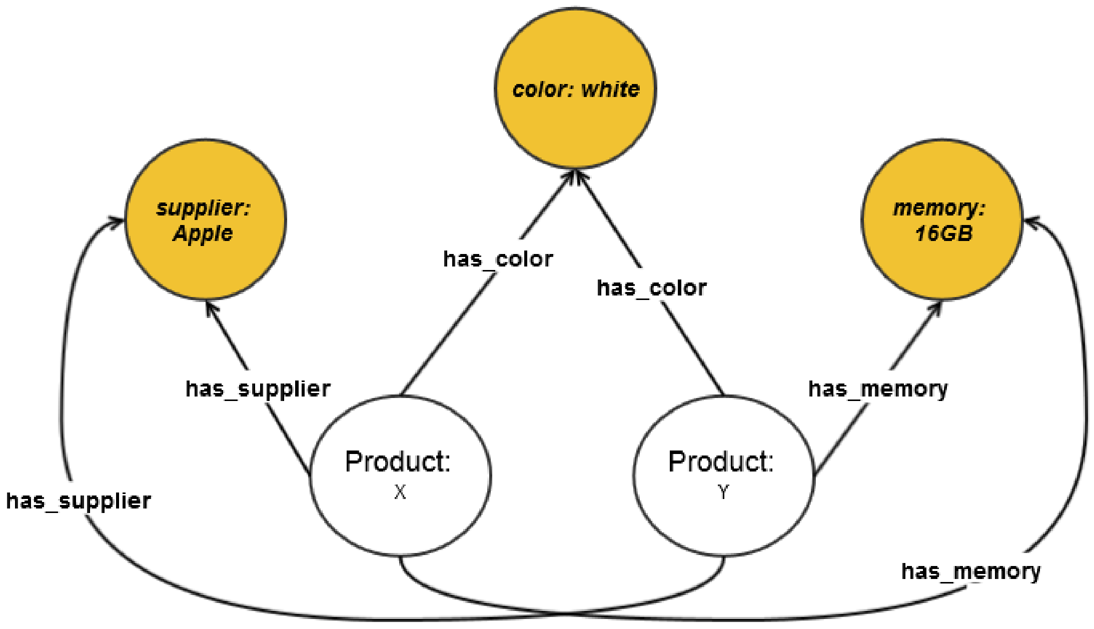


Figure 6.5: Knowledge Graph

What remains is that the answer of how similar these two products. With this type of a structure, it is not that hard to find it. If attributes of X and Y are close to each other then we can say that these are very similar products. However, there is a challenge here; if only one attribute is the same, we cannot say that they are similar products. Similarly in this example, we cannot even determine if they are exactly same products or not. Because for this specific example, one can be an iPad and the other one can be an iPhone. Solution is adding more attributes such as the size of the products in inches, or etc. Obviously, the more attributes, the increased opportunity to find similarities.

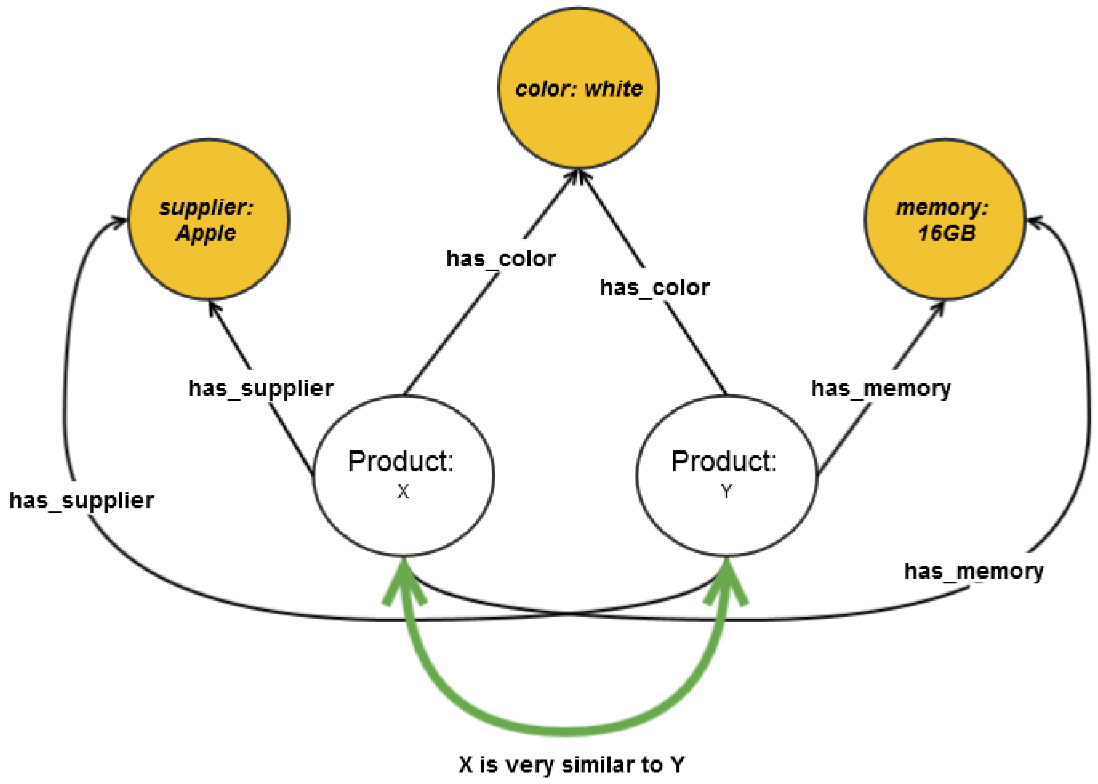


Figure 6.6: Similarity Connection

There is a similarity function f(s) which takes two arguments as inputs, here two products with their connected attributes as lists;

ProductX {*attribute1, attribute2, attribute3, ...*}

ProductY {*attribute1, attribute2, attribute3, ...*}

$$\mathbf{f(s) = (ProductX, ProductY) = ?}$$

At the end, when we can create a link between the products (see Figure 6.6) according to the result of this similarity function **f(s)**, it means we are close to solve issue 1 and 2.

6.5.2.1 Additional Operations for Issue I and Issue II

Building knowledge base, having similarity functions are not enough to solve our issues. There should be additional operations to be performed on the system. Let us recall first two issues;

- Issue I states that when doing categorization, products should be separated from their accessories and listed according to that while displaying search results. How does the system understand which of the products are accessories of other products? This operation should be automatically operated.
- Issue II states that there are more than one vendors who supply the same products to sell customers. How does the system determine if two products from rival vendors are actually the same?

Even though we have bunch of similarity functions and their results, the system cannot be sure about whether the items are accessories of another items (because product names never underline when an item is an accessory), or are they the same products reside in different vendors.

For the first issue, we need to consider an important key point; *if a product is an accessory of another product, its price should be much lower than the other one.* For instance, an iPhone 6 costs around $1074.93, an iPhone 6 EarPods cost $32.25, an iPhone 6 case costs $40.00, an iPhone 6 charger adapter costs $21.47 in Turkey. As you can see the price of iPhone 6 is always higher than its accessories. Therefore checking item prices is a must to deciding the accessories.

For the second issue, we need to consider another key point; *if two products are actually the same, they can be stored in different vendors even though their similarities are close to each other.* For instance, both companies Teknosa and Vatan Computer sell Samsung Galaxy S5 mobile phones. The XML files they sent include textual information about that product. After the system inserting one of them into the database, it should check when iterating over another XML whether that item exists in the database or not. With the help of similarity function, TrendPin decides that these two items are the same. Therefore, it will never create another node for the same product. However, it will also check their suppliers. Even though they are very similar, if their vendors are different they system will know that they are being sold by more than one vendor.

Adding these two additional operations allows TrendPin to overcome first two issues successfully. How about the third issue?

6.5.3 Combining knowledge base with keyword search

Keyword search is the method to find exact match between searched text and the item in the database. TrendPin uses Apache Lucene [5] to search anything. It provides a fast, scalable searching performance as well as incremental indexing and multiple index searching features. When we put our knowledge base on top of Lucene experience, we get better and smarter results.

6.5.3.1 Additional Operations for Issue III

Combining knowledge base with keyword search is not enough to overcome issue III. To recall that, it states there are different product combinations in terms of color, memory, screen size and etc. Some of them are essential when shopping such as memory for specific products. By considering this situation, we added another column to our knowledge base an called it *Impact*. If a category effects the user's choice largely, we set as *"impact of this category is major"*, else we set as *"impact of this category is minor"*.

Category	*Value*	*Impact*
gender	erkek, kadın, unisex, men, women	major
memory	8GB, 16GB, 32GB, 64GB, 128GB, 512GB, 1TB, 2TB	major
color	beyaz, kırmızı, siyah, pembe, bordo, siyah, mavi, mor, turuncu, yeşil, gri, mor, kahverengi, turkuaz, sarı, lacivert, blue, white, black, gold, silver, rose, şeffaf	minor
usingPlace	kol, duvar	major

Table 6.3: A part of TrendPin's domain-knowledge including *Impact* column

Last important point is checking if the user already has the product or not. Usually when a person is searching an accessory of a product, he or she is already have the product, therefore want to search for an accessory of it. When a user sets a product as *"I have it"* in the product's profile page, search field will automatically correct to that sentence into an accessory when trying to search it.

By implementing these features, issue III has been overcame.

6.6 Graph Model Changes

Examples in previous sections were great to explain the techniques in theory, on the other hand they are not efficient to implement on our graph due to the increased complexity foresights. It would require more types of nodes such as color, supplier, memory, and etc., as well as more relationships. Instead, we just created two more nodes labelled with **FeatureOption** and **FeatureGroup**.

A FeatureOption node indicates a single feature to tie it to the products. For instance, *Color: Red, Black, Blue.* Here, Red, Black, and Blue are options, Color is a group. FeatureOption nodes include ids in `Id` as integers, values of options in `Values` as a list of strings, and ids of related FeatureGroups in `FeatureGroupId` as integers.

A FeatureGroup node ties FeatureOptions together. Some features are major and changes the product model, whereas others do not change the model. For example, for iPad color and storage (16 GB, 32 GB, 64 GB) are minor while

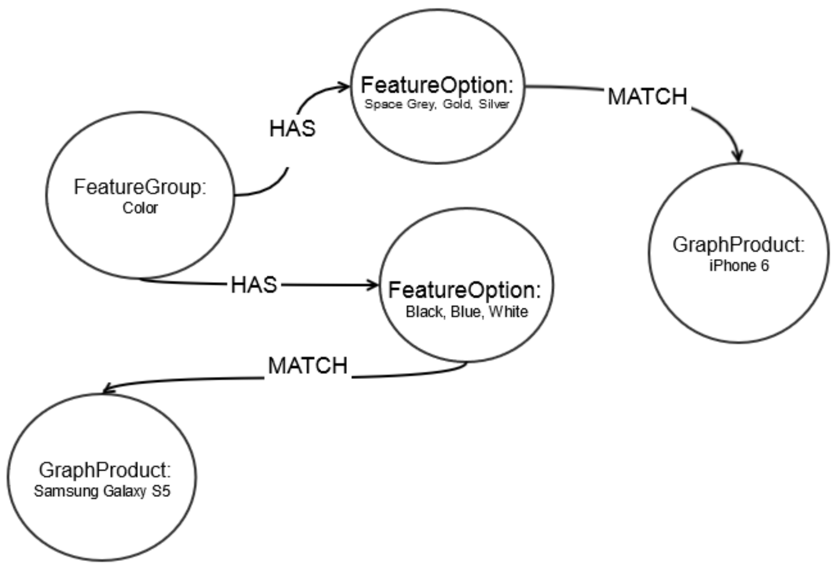


Figure 6.7: Feature extraction in TrendPin

CPU, screen size, memory (512 MB, 1 GB) are major options. The difference is stored within this node in `IsMajor` as boolean formats. Additionally, there are ids of groups in `Id`, and name of the options in `Name` as strings.

These two nodes connected with each other by `HAS` relationships. After that, FeatureOptions are connected to GraphProduct nodes via `MATCH` relationships as in Figure 6.7.

6.7 Effects of Knowledge Based Information Extraction on TrendPin

The effects of knowledge based information extraction can be listed as below;

- TrendPin is now smarter when searching a product. It tries to find out what the user is looking for without even writing the whole name.
- The system decides when two or more products from different vendors are the same and does not create multiple product page of the same product.

- It automatically suggests accessories when the user tries to search for a product if it is already been purchased.
- It creates an advanced searching page containing all relevant features about that specific product with check boxes after a product has been searched for the first time.

6.8 Challenges and Drawbacks of Feature Extraction Methodology

It is inevitably true that feature extraction and information retrieval techniques are very good at what they intent to do. Especially information extraction is the perfect way to provide support for the knowledge identification in an automatic way. However, just like every concept, these techniques have some challenges in practice.

6.8.1 Challenges

Major challenges of the topic are described by Ciravegna in 2001[44]. He claims that there are two distinct challenges;

- Automatic adaptation to different text types.
- Human-centred issues in copying with real users.

He explains that the first challenge comes from four tasks of information extraction such as adaptation of new domain information, adaptation of distinct sub-languages, adaptation of different text genres, and adaptation of different document types. is developing methodologies able to fill the gap between the two approaches in order to cope with different text types. These tasks create very serious limitation for portability. He also adds that they are not able to cope with the variety of extralinguistic structures (e.g. HTML tags, and document formatting)

For the second challenge, he underlines that information extraction is related to human-computer interaction during an application development process. Users does not need to be expert on the area, so that they need to be supported during the whole adaptation process to maximize effectiveness of the final application.

Apart from those above, Grishman (1997)[59] discusses three design issues on the topic;

- To parse or not to parse
- Portability
- Improving performance

6.8.2 Drawbacks

Feature extraction is very useful when it is applied reasonable and well. On the other hand, according to some researchers, there are a number of drawbacks of using this methodology on the knowledge management systems.

Manolescu (1998)[75] sums up those drawbacks in his article;

- When a new item is inserted, there will be additional processes.
- Feature extraction function is not easy to determine.
- Indexing methods are not always scale well.
- There may be additional storage for the features.

In addition, Guyon and Elisseeff (2006)[61] described more problems and required needs of the topic;

- More theoretically grounded algorithms.
- Better estimation of the computational burden.
- Better performance assessment of feature selection.

More research should be done on this area to be able to optimize the methods and get better performance.

6.9 Summary

Graph databases are very powerful and graph designs can be extended in every direction. There is not a single design decision for every graph application. They are only limited with imagination of developers. Therefore, we show that information extraction and its methods can also be applied on graph models.

In this chapter, we explained information extraction and its methods. We also presented problems originated from graph design of TrendPin and related information extraction concept with those problems to find a suitable solution. Then, we implemented a knowledge base for TrendPin and explained implementation process step by step.

Graph model of TrendPin changed and we illustrated these changes on the graph. At the end we listed the effects of feature extraction processes on the system as well ass stating challenges and drawbacks of such methods. In the next chapter, we compare relational and graph model using queries and explain results.

Chapter 7

Tests

In this chapter, we discuss the differences of both relational model and graph models in terms of design challenges and performances via experimental query results. Note that the main purpose of this thesis work is to provide a comparison between database management systems. Feature extraction implementation is applied as a result of graph model design. Therefore, we do not state a comparison of feature extraction processes between relational model and graph model and exclude it out from this section.

7.1 Model Comparisons

Querying to expect same result sets on both relational and graph model is of course the best way to compare two versions of the same application. Simply, which one operates faster is going to be treated as a better design choice. However, choosing the best fitting technique usually depends on application domain, time required to complete the project, and developers experiences. For instance, when developers experience is low on graph databases, there would be a required training. If the time is very crucial to deliver the system to the customers, it would be very difficult to choose graph databases over relational models., because relational models are already in the market for many years and developers experience would be much higher. Moreover, some applications may not be suitable to design as graphs, in this case other models can be chosen by database developers.

We divide comparisons into two; experimental query results to measure performances in a timely manner, and remarks on design challenges we face during the design process of both models in the next section.

7.1.1 Experimental Query Results

We compared two distinct models of TrendPin by executing SQL and Cypher queries for the same result sets.

We measured running times of those queries in order to show which database performs better on which operations. We have only tested with reading queries, since it is believed to be more challenging to retrieve data rather than writing in terms of velocity.

We ran each query 500 times on a PC which had Windows 8.1 64 bit operating system, 16 GB ram, Intel Core i7-4700 CPU at 2.40 GHz processor.

For relational model, we used Microsoft SQL Server 2012 database product. For graph model, we used Neo4j 2.1.7 Community Edition.

Experiments include description of the experiment, SQL query, Cypher query, and the query results in the form of bar charts with average of all trials.

7.1.1.1 Experiment 1

For the first experiment, we try to retrieve many results and compare how well two models handle such an operation. To analyse this situation, we want to *"find the list of member friends of users for the given UserId"*. Note that some users may not be members of the system, therefore we exclude them here.

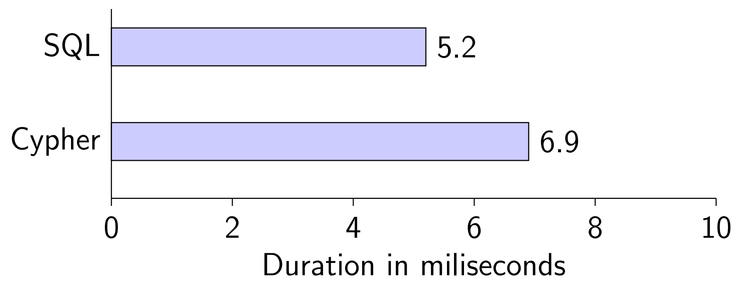


Figure 7.1: Average results for experiment 1

SQL query for this experiment is given below;

```
SELECT f.UserId
FROM Follower f
WHERE f.FollowerId = "UserId"
UNION ALL
SELECT f.FollowerId
FROM Follower f
WHERE f.UserId = "UserId"
```

Cypher query for this experiment is given below;

```
MATCH (u:GraphUser) -[:IS_FRIEND_OF]-> (f:GraphUser)
WHERE u.Id = "UserId" AND f.IsMember = true
RETURN f, SKIP 0, LIMIT 100
```

7.1.1.2 Experiment 2

For the second experiment, we try to retrieve results from single type of matchings and compare how well two models handle such an operation. This is easy for both models due to the fact that there is not any join operations needed. To analyse this situation, we want to *"find first five products a user has ever selected as **BUYING**"*.

SQL query for this experiment is given below;

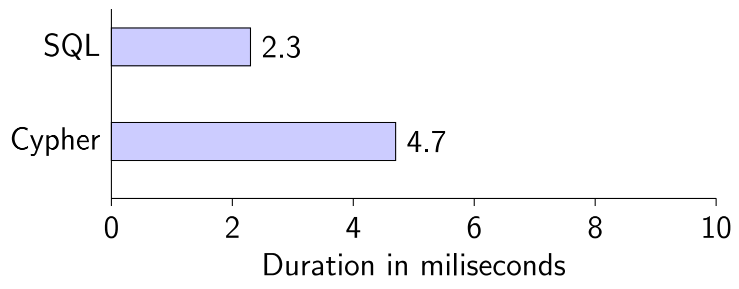


Figure 7.2: Average results for experiment 2

```
SELECT TOP(5)
FROM User u, Product p, UserProductRelation upr,
     RelationType rt
WHERE u.Id = upr.UserId AND
      upr.RelationId = rt.Id AND
      upr.ProductId = p.Id AND
      rt.Type = "BUYING"
ORDER BY upr.CreationDate ASC
```

Cypher query for this experiment is given below;

```
MATCH (u:GraphUser) -[rel:BUYING]-> (p:GraphProduct)
WHERE u.Id = "UserId"
WITH p as prod
ORDER BY rel.Date ASC
LIMIT 5
RETURN prod
```

7.1.1.3 Experiment 3

For the third experiment, we try to retrieve results from multiple type of matchings and compare how well two models handle such an operation. Now, we include join operations for relational model and two times match operation for graph model. To analyse this situation, we want to *"find the shops of last two products which a user selected as* ***NEEDSHELP***.

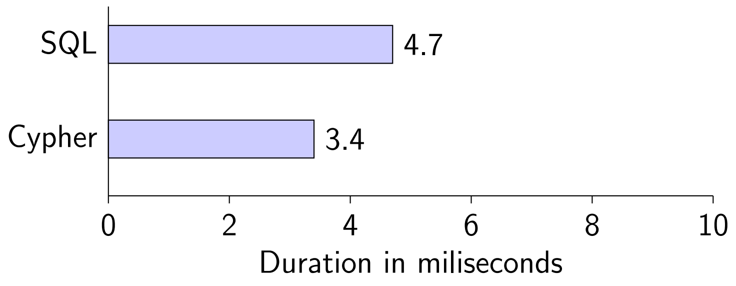


Figure 7.3: Average results for experiment 3

SQL query for this experiment is given below;

```
SELECT sh1.Name, sh2.Name
FROM Shop sh1, Shop sh2, Product p
WHERE sh1.Id = p.ShopId AND
      s1.Id = s2.Id AND
      p.ProductId IN
    (SELECT TOP (2) prod.ProductId
     FROM User u, Product P, UserProductRelation upr,
          RelationType rt
     WHERE u.Id = upr.UserId AND
           upr.RelationId = rt.Id AND
           upr.ProductId = p.Id AND
           rt.Type = "NEEDSHELP"
     ORDER BY upr.CreationDate DESC)
```

Cypher query for this experiment is given below;

```
MATCH (u:GraphUser) -[rel:NEEDSHELP]-> (p:GraphProduct)
WHERE u.Id = "UserId"
WITH prod
ORDER BY rel.Date DESC
LIMIT 2
MATCH (prod) <-[:HAS]- (s:GraphShop)
RETURN s.Name
```

7.1.1.4 Experiment 4

For the fourth experiment, we try to retrieve posts of users and compare how well two models handle such an operation. To analyse this situation, we want to *"find a post of a user given by post id about a product"*.

SQL query for this experiment is given below;

```
SELECT p.Id, p.UserId, p.ProductId
       p.Text, p.FormattedText, p.LikeCount,
       p.CommentCount, p.Date,
       u.FirstName, u.LastName, u.Email
FROM User u, Post p, Product prod
WHERE u.Id = p.UserId AND
      p.ProductId = prod.Id AND
      p.Id = "PostId"
```

Cypher query for this experiment is given below;

```
MATCH (u:GraphUser)-[r:POSTS]->(p:GraphPost)-
    [:FOR]->(prod:GraphProduct)
WHERE p.Id = postId
RETURN p, r, u, prod
```

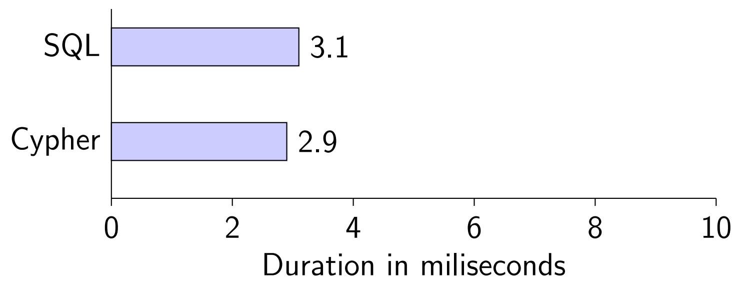


Figure 7.4: Average results for experiment 4

7.1.1.5 Results

- In experiment 1, we wanted to show how a UNION statement in SQL query performs against Cypher version. According to the average results, SQL performs better than Cypher.
- In experiment 2, we wanted to see how multiple join operations race against Cypher equivalence. It appears that Cypher is a bit slower, in contrast to SQL performance.
- In experiment 3, we wanted to measure how sub queries of SQL with many tables that join together performs with Cypher. Here, Cypher performs better against a complex query of SQL.
- In experiment 4, we wanted to retrieve specific results with very straightforward queries. Performances of both queries are very close in these trials.

As a result of all the experiments we have conducted, neither relational databases nor graph databases are best fit for each datasets and queries. They did not perform identical on different queries, therefore it is not possible to say that one of them is better. Eventually, it can be said that application domain, experience of developers, and querying style are significant factors when deciding which data model should be used for projects.

7.1.2 Design Challenges

7.1.2.1 Maturity

Databases are not created immediately after requirements are collected. In order to model a database, requirements should be interpreted in detail. A good design is only possible with well understood requirements and application domains. Problems arising from the design is hard to recover when implementation of the system has begun. Thus, a model should be robust and stable enough to cover all requirements.

Relational databases are old trends. For more than 40 years, a lot of relational models created from a lot of different developers. It is not always easy to decide how to partition tables into two or more. It is up to developers choice and the experience in the topic. Relational models require normalization operations to hold for whole database design. The community of relational databases and SQL includes tons of developers who write on blogs, create slides, share experiences and examples on the internet. When someone stuck on an issue, it is always possible to find a solution idea online. For this reason, relational model design is more mature and advantageous.

On the other hand, graph databases are a lot newer in the market. There are a lot of products and different programming languages for each. It is not similar to *"learn SQL and use most of the products"* mentality. Therefore, it is not as easy as finding well prepared documents about graph databases. There are less people in the community to get supported. Neo4j is far the best to provide developer manuals, provide a stable product, and continuously improve features and issues regarding to the environment among others. As a result, choosing Neo4j is a good move and it is more mature than other products, however it is still not as easy to find solutions when stuck on a problem as relational models in the community. Since graph databases attract many attentions, this situation will soon be changed.

7.1.2.2 Resilience

Inserting, updating and deleting operations do not effect the design conditions of databases. However, when a relational schema is decided, it is not possible to go beyond this agreed schema. With the addition of new requirements, it may be impossible to change columns, add new attributes since it is probably going to violate all normalization conditions on other tables. It may be complicated to decide what to do in such situations. Hence, relational models are hard to adopt changes over time and their resilience is limited.

Graph databases provide easy extensions for new coming data. New labels on nodes and new relationships do not effect other nodes in the database. It is as easy as simulating a graph on a board and implement changes on graph without intervening other entities. Moreover, lacking a scheme allows converting all requirements into entity and relationship pairs in many different methods. In chapter 6, we implemented feature extraction addition on the graph and overall design we provided in chapter 5 did not changed at all. For these reasons, graph databases are more resilient than relational models for sure.

7.2 Summary

In this chapter, we presented experiments we conducted on TrendPin in terms of querying with two different database technologies. We show that both databases perform different towards distinct experiments. In some trials, Microsoft SQL Server performed better, whereas in others, Neo4j produced faster result sets. Therefore, it is not a valid statement that one type of database always performs better. It is up the to problem domain, design of the models, and query contents.

We also discussed both models from maturity and resilience perspectives. We stated Microsoft SQL Server is more mature than Neo4j. However, Neo4j is still more resilient and flexible.

Chapter 8

Conclusion

8.1 Summary

The reason behind of upgrading systems is that previous versions are not able to meet new requirements. These requirements are often system specific. In general developers want their system to be used easily, work as efficient as it can, handle related jobs without any problem. Researchers are aware of Big Data challenges; "Today we are witnessing an exponential growth in the volume and detail of data captured by enterprises, the rise of multimedia, social media and online social networks, and the Internet of Things" (Dobre and Xhafa, 2013) [55]. Relational Database Management Systems can no longer help to solve the problem of Big Data. Thus they have issues when it comes to manage and analyse it efficiently. To remind from previous chapter, when the number of users and interactions are increased, it becomes very complicated for the system to keep itself stable and work efficient at the same time.

Big Data is a huge area to be researched. Many researchers are agreed on the solutions are lies within NoSQL. Moniruzzaman and Hossain (2013) [76] underlines that there are different alternatives to manage Big Data such as NewSQL and NoSQL instead of Relational Database Management Systems. Hecht and

Jablonski (2011)[65] agrees on this by stating that ”many companies and organizations developed own storage systems, which are now classified as NoSQL databases”.

In this thesis study, we have discussed Big Data extensively from its features, to sources and challenges. We have stated that they require additional operations to work better. We have explained how MapReduce procedure processes and why Hadoop Distributed File System is a pioneer of new solutions for Big Data applications. We have separated database management systems into two as relational databases and NoSQL trends and explained both approaches in detail. We discussed nearly every single details of relational tables, ACID properties and normalization techniques and discussed SQL. Moreover, we discussed each NoSQL technologies also by providing example products and talked which is best for which application domain.

We chose graph databases to design for the rest of this study and explained how graph designs can be implemented by using Neo4j. We introduce Cypher query language for graph data handling with examples. However there was still an important question remained to be discussed; *“How well are graph databases when compared to relational database models in terms of performance, scalability, and other Big Data needs?”*. In order to answer this question, we have introduced a system called TrendPin.

8.2 Contributions

Contributions of this study can be summarized as follows:

- We discussed different database technologies and present an comparative analysis about NoSQL trends, especially graph databases with Neo4j product and Cypher query language in Chapter 3.
- We proposed TrendPin and explained two different database models on the system, relational model with Microsoft SQL Server, and graph model with

Neo4j in Chapter 5.

- We discussed information extraction techniques and implemented a knowledge base for TrendPin to find answers to the issues of its graph model in Chapter 6.
- We present experimental results with two distinct models in terms of query comparisons, as well as design challenges in Chapter 7.

8.3 Drawbacks

Drawbacks of this study can be summarized as follows:

- In this study we aimed to cover only two type of database management systems. As we have discussed before, there are more types and a large number of different products even for the types we selected to experiment on. Therefore, it is not possible to agree one of them is the best for now.
- We could not try queries on feature extraction entities which we covered in Chapter 6, due to the fact that there was not an implementation of such a technique in relational model. It would be comparable if implementation was completed in relational model too.
- We experimented with only four different types of queries. There could be written more and compare more statements in both models.

8.4 Future Work

Future work of this study can be summarized as follows:

- TrendPin never went live fully functional. We tested the system to find bugs, features where improvements needed, and etc. such as a beta testing. After the website is online, more challenges will probably follow.

- We would like to develop more databases for the same application such as MongoDB for document databases, or Cassandra for column-family/wide column databases and compare same queries which we have covered in this thesis to extend our study.
- We would like to improve our knowledge base to provide much more smarter results for the users of this system.

BIBLIOGRAPHY

[1] Accumulo. https://accumulo.apache.org/.

[2] Aerospike. http://www.aerospike.com/.

[3] Allegrograph. franz.com/agraph/allegrograph/.

[4] Ambari. ambari.apache.org/.

[5] Apache lucene. http://lucene.apache.org/core/.

[6] Brightstardb. https://brightstardb.com/.

[7] Cassandra. http://cassandra.apache.org/.

[8] Couchdb. couchdb.apache.org/.

[9] Data never sleeps. http://www.domo.com/learn/infographic-data-never-sleeps/.

[10] Data storage for modern high-performance business applications. http://msdn.microsoft.com/en-us/library/dn313285.aspx.

[11] Dynamo. http://aws.amazon.com/dynamodb/.

[12] Elasticsearch. http://www.elasticsearch.org/.

[13] Hbase. http://hbase.apache.org/.

[14] Hive. hive.apache.org/.

[15] Hypertable. http://hypertable.org/.

[16] Infogrid. infogrid.org/.

[17] Jasdb. www.oberasoftware.com/jasdb-2/.

[18] Mongodb. http://www.mongodb.org/.

[19] Neo4j. www.neo4j.org/.

[20] Online retail payments forecast 2010 2014: Alternative payments growth strong but credit card projected for comeback. https://www.javelinstrategy.com/Brochure-171.

[21] Open information extraction. http://ai.cs.washington.edu/projects/open-information-extraction.

[22] Pig. https://pig.apache.org/.

[23] Ravendb. ravendb.net/.

[24] Redis. http://redis.io/.

[25] Riak. http://basho.com/riak/.

[26] Simpledb. http://aws.amazon.com/simpledb/.

[27] Thrudb. code.google.com/p/thrudb/.

[28] Titandb. thinkaurelius.github.io/titan/.

[29] Trinity. http://research.microsoft.com/en-us/projects/trinity/.

[30] Voldemort. http://www.project-voldemort.com/voldemort/.

[31] Windows azure storage. http://www.windowsazure.com/en-us/documentation/services/storage/.

[32] Zookeeper. zookeeper.apache.org/.

[33] Ibm big data success stories. http://public.dhe.ibm.com/software/data/sw-library/big-data/ibm-big-data-success.pdf, 2011.

[34] Ali M. Al-Khouri. Privacy in the age of big data: Exploring the role of modern identity management systems. *World Journal of Social Science*, 1, 2014.

[35] Cloud Security Alliance. Top ten big data security and privacy challenges. https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Top_Ten_v1.pdf, November 2012.

[36] Abhinay B. Angadi, Akshata B. Angadi, and Karuna C. Gull. Growth of new databases & analysis of nosql datastores.

[37] María del Pilar Angeles and Victor González Castro. V+ h: Hybrid architecture for dss and oltp. *International Journal of Information Management*, 33(6):940–947, 2013.

[38] Paolo Atzeni, Giorgio Orsi, Christian S. Jensen, Sudha Ram, Letizia Tanca, and Riccardo Torlone. The relational model is dead, sql is dead,... and i don't feel so good myself. 2012.

[39] Otakar Borůvka. O jistém problému minimálním (about a certain minimal problem).

[40] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[41] Mike Buerli. The current state of graph databases. 2012.

[42] Donald D Chamberlin and Raymond F Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264. ACM, 1974.

[43] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *OSDI 2006 Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, 7:15, 2006.

[44] Fabio Ciravegna. Challenges in information extraction from text for knowledge management.

[45] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[46] Edgar F. Codd. Normalized data base structure: A brief tutorial. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 1–17. ACM, 1971.

[47] Edgar F. Codd. *Recent Investigations in Relational Data Base Systems.* IBM Thomas J. Watson Research Division, 1974.

[48] Datameer. http://www.datameer.com/product/hadoop.html.

[49] Datastax. Nosql in the enterprise, a guide for technology leaders and decision-makerse. http://www.datastax.com/wp-content/uploads/2011/09/WP-DataStax-NoSQL.pdf, October 2013.

[50] Jeffrey Dean and Sanjay Ghemawat. Map-reduce: Simplified data processing on large clusters 0018-9162/95. *D OSDI IEEE*, 2004.

[51] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, volume 7, pages 205–220, 2007.

[52] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 international conference on Management of data*, pages 1209–1220. ACM, 2013.

[53] Francis X. Diebold. "big data" dynamic factor models for macroeconomic measurement and forecasting. 2000.

[54] Jean-Pierre Dijcks. Oracle: Big data for the enterprise. http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf, June 2013.

[55] Ciprian Dobre and Fatos Xhafa. Intelligent services for big data science. *Future Generation Computer Systems*, 2013.

[56] Edd Dumbill. What is big data? an introduction to the big data landscape. *oreilly. com, http://radar. oreilly. com/2012/01/what-is-big-data. html*, 2012.

[57] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, 2(3):262–278, 1977.

[58] Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *arXiv preprint arXiv:1308.1479*, 2013.

[59] Ralph Grishman. Information extraction: Techniques and challenges. In *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, pages 10–27. Springer, 1997.

[60] Rong Gu, Xiaoliang Yang, Jinshuang Yan, Yuanhao Sun, Bing Wang, Chunfeng Yuan, and Yihua Huang. Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters. *Journal of Parallel and Distributed Computing*, 74(3):2166–2179, 2014.

[61] Isabelle Guyon and André Elisseeff. An introduction to feature extraction. In *Feature Extraction*, pages 1–25. Springer, 2006.

[62] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.

[63] Frank Harary and Edgar M Palmer. Graphical enumeration. Technical report, DTIC Document, 1973.

[64] Jonathan Hausmann. http://autoinflammatorydiseases.org/social-media-2/physicians-guide-social-media-part/.

[65] Robin Hecht and Stefan Jablonski. Nosql evaluation. 2011.

[66] Jerry R. Hobbs and Ellen Riloff. Information extraction. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing,*

Second Edition. CRC Press, Taylor and Francis Group, Boca Raton, FL, 2010. ISBN 978-1420085921.

[67] Mr Mahesh G. Huddar and Manjula M. Ramannavar. A survey on big data analytical tools. *International Journal of Latest Trends in Engineering and Technology (IJLTET)*.

[68] Jing Jiang. Information extraction from text. In *Mining text data*, pages 11–41. Springer, 2012.

[69] M. Kiran, Amresh Kumar, Saikat Mukherjee, and G. Ravi Prakash. Verification and validation of mapreduce program model for parallel support vector machine algorithm on hadoop cluster. 2013.

[70] Douglas Laney. 3-d data management: Controlling data volume, velocity and variety. *META Group Research Note, February*, 6, 2001.

[71] Lynn Langit. Hadoop mapreduce fundamentals. http://www.slideshare.net/lynnlangit/hadoop-mapreduce-fundamentals-21427224.

[72] R. Duncan Luce and Albert D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[73] Prabath Maduranga. http://www.prabathsl.com/2013/02/document-oriented-database_14.html.

[74] Markus Maier. Towards a big data reference architecture, 2013.

[75] Dragos-Anton Manolescu. Feature extraction: A pattern for information retrieval. *Proceedings of the 5th Pattern Languages of Programming, Monticello, Illinois*, 1998.

[76] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *International Journal of Database Theory & Application*, 6(4), 2013.

[77] Cory Nance, Travis Losser, Reenu Iype, and Gary Harmon. Nosql vs rdbms-why there is room for both. *Proceedings of the Southern Association for Information Systems Conference, Savannah, GA, USA.*

[78] Claire Nédellec and Adeline Nazarenko. Ontologies and information extraction. *arXiv preprint cs/0609137*, 2006.

[79] P. Srinivasa Rao, K. Thammi Reddy, and MHM Krishna Prasad. A novel and efficient method for protecting internet usage from unauthorized access using map reduce. *International Journal of Information Technology and Computer Science (IJITCS)*, 5(3):49, 2013.

[80] Akshay K. Singh. Performance isolation in cloud storage systems. 2013.

[81] Jerry A. Smith. Field note: What makes big data big some mathematics behind its quantification, data scientist insights. 2012.

[82] Carlo Strozzi. Nosql - a relational database management system. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%2520%20page/, 2007-2010.

[83] Shan Suthaharan. Big data classification: Problems and challenges in network intrusion prediction with machine learning. In *Big Data Analytics workshop, in conjunction with ACM Sigmetrics*, 2013.

[84] Raoul-Gabriel Urma and Alan Mycroft. Source-code queries with graph databases - with application to programming language usage and evolution. *Science of Computer Programming*, 2013.

[85] Tom White. *Hadoop: The definitive guide.* O'Reilly Media, Inc., 2010.

[86] Fei Wu and Daniel S Weld. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127. Association for Computational Linguistics, 2010.

[87] Jean Yan. Big data, bigger opportunities. 2013.