Profiling Data and Beyond: Gaining Insights from Metadata

Inauguraldissertation zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften durch die Wirtschaftswissenschaftliche Fakultät der Westfälischen Wilhelms-Universität Münster

Vorgelegt von

Fabian Schomm-von Auenmüller, MSc
aus Viersen

Mai 2018

Dekanin	Prof. Dr. Theresia Theurl
Erster Gutachter	Prof. Dr. Gottfried Vossen
Zweiter Gutachter	Prof. Dr. Stephan Meisel
Mündliche Prüfung	06.07.2018

Contents

List of Figures

List of Tables

1 Introduction

The rate at which data is generated increases every day [35], and at the same time, the cost for data storage is continually decreasing [15]. This development allows many organizations to store vast amounts of data with little regard as to what should be done with it. The raw data however has no inherent value by itself, but it represents a source of untapped potential.

It has been said that "data is the new oil" [24]. This analogy does not refer to the scarcity or the non-renewability of the resource, but rather to the fact that it needs to be processed and refined to gain value from it. In this vein, uninterpreted data corresponds to crude oil, and refining it involves gathering knowledge about it, putting it into context, and analyzing it. A refinement of crude oil into usable products such as gas, plastic, or other chemicals is analogous to a transformation of data into *information* [Ack89, LLS10]. This transformation process is accomplished through an interpretation of the data. In other words, i. e., information is interpreted data [AN95, p. 197].

Having correct and relevant information is crucial in all organizations as it is the basis for any kind of decision making. However, the sheer volume of data that is available makes it challenging to select the relevant parts and interpret them properly. To facilitate this issue, it can be helpful to preprocess the data and extract metadata about it. Metadata is "data about data" [HT03, p. 9] and can be used to create an easily-digestible overview of the data it describes. The types of metadata range from simple counts of rows or null values over value distributions and outliers to complex integrity constraints such as foreign key dependencies. The act of extracting metadata is commonly referred to as *data profiling* and the resulting set of metadata is called the *data profile* [AGN15].

Data profiling has the potential to save money and time in data-driven scenarios. For example, in any business intelligence project, a data preparation phase is recommended to ensure that the input data is consistent and accurate before it is passed into analysis. This data preparation phase can take a substantial amount of time: In a survey conducted by TDWI in 2016, it was reported that 73% of respondents spent at least 41% of their total time in an analytics project on preparing data compared to the time spent performing analysis [Sto16, p. 21]. In the same survey, 86% of respondents said that it is important to reduce the amount of time and resources spent on data preparation [Sto16, p. 22]. Similar numbers are reported by IBM, who estimate that 70% of the time in an analytics project is spent on cleaning and integrating data [TSRC15]. Data profiling is a crucial part of data preparation, because it enables a user to assess the data quality and spot inconsistencies. Thus, it is expected that a better understanding of data profiling can help with facilitating data preparation.

1.1 Aims of this Thesis

Data profiling has been the subject of much research work, but it is still not considered to be a research area in its own right [Nau13]. Instead, it is often treated as a byproduct of other areas, such as data quality [Ols03], data cleaning [RD00], or data mining [Pyl99]. This has led to a situation in which data profiling is admittedly recognized as an important and valuable discipline, but it does not have a common body of knowledge that establishes a set of agreed-upon and generally accepted methods and terms. Thus, the first part of this thesis aims to:

> *Establish a definition of data profiling and provide an overview of its constituent components*

Most of the existing work on data profiling considers only the dataset itself as a source for metadata that describes its content. This is a limited view that restricts what types of metadata can be gathered. For example, the provenance of data, i. e., its origin and transformation history [BKWC01], is often neglected in data profiling, but knowing where data comes from is very useful, e. g., when tracking down errors. There is often an interesting reciprocal relation between the data and the processes and applications that interact with it. Including this relation during profiling can potentially lead to many more useful insights that would otherwise go unnoticed. The second part of this thesis therefore aims to go *beyond* the data and include its context into the considerations:

> *Expand profiling and its methods to the context of data*

This will yield a more holistic view of a dataset, which can be applied in a wide range of scenarios and domains.

1.2 Structure of this Thesis

This thesis is subdivided into eight chapters. After this introduction, the term *profiling*, detached from any data, is examined closely in Chapter 2. To this end, its linguistic background is analyzed and an overview of commonly encountered application areas is given. The shared similarities are compiled into a generic model that captures the essence of what profiling is. This is followed by Chapter 3, which introduces a characterization of the problems to which data profiling can be a solution. This considers types of data that may be encountered, typical tasks that revolve around data, and who the users that face these tasks are. The chapter closes with insights into what is commonly done in cases where data profiling is not considered as the method of choice.

Chapter 4 presents the current state of the art of data profiling and addresses the first research aim. It establishes a definition of the term and provides a detailed list of individual metadata types that make up a data profile as well as a classification scheme for them. This chapter is completed by a short overview of software tools and research projects that are used in the profiling context. Subsequently, Chapter 5 assumes an organizational perspective and characterizes six different data profiling variants along the dimensions *purpose, input,*

executor, method, output, user, and *application areas*. This provides a comprehensive summary of when, why, and by whom profiling should be considered and concludes the first part of this thesis.

The second part of this thesis addresses the second research aim. To this end, Chapter 6 first broadens the scope of metadata to those types that are not intrinsically available from the data itself, but must be sourced from external sources. A list of these extrinsic metadata types is provided and complemented by a discussion of their sources and a process for extraction. Then, Chapter 7 shifts the focus away from the data by introducing an abstract model for the context of data, which consists of processes and applications. This model is used to envision profiling methods that can be applied to this extended context. The application of the model is discussed alongside three different practical examples.

Finally, this thesis is concluded in Chapter 8 with a summary of the main contributions and an outlook on further research opportunities.

Note that throughout this thesis, gender-specific terms may be used in an effort to ease the flow of reading. These should be understood as referring to both genders, unless explicitly stated otherwise.

1.3 Running Scenario: The CloudHost Case

In the following chapters, numerous different concepts will be explained and illustrated. In an effort to facilitate their understanding and demonstrate the practical applicability, examples are used whenever feasible. Most of these examples will be taken from a fictitious case about a company called CloudHost.

CloudHost is an IT company that offers a cloud platform for clients to rent and host arbitrary software on. There are several hundred clients who use this platform to run thousands of different programs, applications and services, which are either third-party software or implemented by CloudHost themselves. CloudHost records a wide variety of data about which client is running which software, in which version and configuration. Most of this data is stored in a relational database management system, but CloudHost also employs a document-oriented database which stores semi-structured data for use cases where a higher degree of flexibility is required (e. g., web traffic data).

To expand their business, CloudHost employs a dedicated team of sales representatives that regularly visit prospective and existing clients to inform them about the latest product offerings in a sales pitch. In order to improve the quality of these pitches, the sales team needs detailed information about the client, his priorities, preferences, and purchase history at CloudHost. This enables a custom-tailored and highly focused negotiation process, which is highly beneficial for all involved parties. However, in order to provide these client information to the sales team, numerous data sources need to be integrated and analyzed first. This is the core challenge of the CloudHost case, and it is addressed with the use of data profiling techniques.

The inspiration for this example scenario came from a real project that has been conducted by the Databases and Information Systems Group in corporation with an undisclosed company from Münster. This project started in October 2014 and concluded in August 2015. To comply

with non-disclosure agreements, the name and any identifiable references are obfuscated and replaced with fictional ones, while preserving as much of the project and the involved challenges as possible.

Part I

State of the Art

2 Profiling Fundamentals

This chapter introduces the fundamentals of profiling by first defining the term in Section 2.1. Then, the most common application areas that include the term *profiling* are considered and their differences and similarities highlighted in Section 2.2. The results are used in Section 2.3 to develop a generic profiling model that shows the inputs and outputs of profiling as well as the activities in between. Lastly, Section 2.4 arranges the profiling activities with respect to their main purpose, which can range from description over prediction to prescription.

2.1 Definition

In order to fully understand what the term profiling means, its linguistic characteristics and etymology are examined first. The term *profiling* is composed of the verb *to profile* and the suffix *-ing*. The first part, to profile, has its origin in the Italian word *profilare*, which means "to represent in profile" and goes back to the 1650s [11]. This indicates that the verb's meaning is dependent on the noun *profile*. Looking at this noun, its roots can be traced back to the Italian word *profilo*, meaning "a drawing in outline" [11]. Drawing something in outline involves marking only the contours of a shape or figure while omitting all details such as shading and colors. The purpose of an outline is to visually represent the most important characteristics of an object.

So far, there is no further specification of the object, the *something*, that is being profiled or outlined. However, looking at more modern definitions of the word yields more insight: The online dictionary at Dictionary.com defines *to profile* as "to draw a profile of", which is very similar to the Italian origin described above. The noun *profile*, however, is defined as "(1) the outline or contour of the human face, (2) a picture or representation of the side view of a head, or (3) an outlined view, as of a city or mountain" [5]. Here, the object to be profiled is specified to be a human face, or a city/mountain, which hints at concrete application areas in which profiling is performed.

Examples for this notion of *profile* can be seen in Figure 2.1. Figure 2.1a shows the silhouettes of Münster's most iconic buildings. The minimalistic style omits many details, such as color or material, and does not even correctly reflect the height or position of the buildings. Still, this skyline is sufficient to enable a beholder that knows about Münster to uniquely and correctly identify the depicted city. As another example, Figure 2.1b shows the silhouette of a man in black and white. Theoretically this could be any man, but due to the high recognition value, it is obvious that this is a profile of Donald Trump. This depiction makes use of the unique and characteristic features of his appearance (most notably the shape of his hair) to give a short and compact visual representation. In a sense, a profile shares some similarities to a model, which is an abstraction of a real-world object that highlights only the important parts [Sta73].

(a) Skyline of Münster. Source: [33].

(b) Silhouette of Donald Trump. Source: [29].

Figure 2.1: Examples of different types of profiles.

The second part of the word *profiling*, "-ing", transforms the verb into a gerund, which has two effects: Firstly, it signifies an ongoing process, i. e., something that is *being* done. As such, it is implied that there is no fixed point when profiling ends. Secondly, it emphasizes the fact that *profile* is to be understood as a verb, and should not be confused with the identically written noun.

Grammatically speaking, the combination of "to profile" and "-ing" into *profiling* should not change the semantic meaning and only indicates that profiling is being done. However, the historic and linguistic usage has transcended *profiling* into much more than that, which is reflected in several definitions. For example, the following definition for profiling can be found on Wikipedia: "Profiling is the extrapolation of information about something, based on known qualities" [43]. The Merriam Webster dictionary offers a similar definition: "the act or process of extrapolating information about a person based on known traits or tendencies" [19]. The most striking difference is that *profiling* is described as *extrapolating*, as opposed to *representing* in the case of *to profile*. Extrapolating means to derive something previously unknown, possibly with the use of inference and reasoning rules. In this sense, profiling is concerned with the creation of knowledge about a given object whereas a profile focuses on reduction and abstraction to provide a concise summary of an object and its properties. However, some application areas use the word *profiling* to not only describe the act of profile creation, but also the usage and application of existing profiles in a given setting. To account for these cases and conclude this section, the so-far discussed points are summarized in the following working statement:

> "Profiling is the act of creating a profile of a given input object based on observation, analysis, and extrapolation. Additionally, profiling also relates to the application and exploitation of profiles in a given setting."

2.2 Application Areas

The term *profiling* most often occurs in combination with another word, which usually specifies the object that is being profiled. Thus, those different types of profiling denote various application areas. This section explores these application areas that are most frequently encountered in order to show the breadth of the field and identify similarities and differences.

Each subsection is concluded with a summary of the most important characteristics of the respective application area.

2.2.1 Offender Profiling

Offender profiling is the art of creating profiles of suspects for a criminal act in order to help law enforcement to unequivocally identify (and ultimately catch) the perpetrator of a committed crime. Woodhams described it as "the inference of offender characteristics from offense behaviors" [WT07]. To this end, many different characteristics of a criminal are considered such as his physical and emotional state, his behavior and his psyche [Tur11].

Due to this breadth of information sources offender profiling is a multidisciplinary approach that draws from the insights of multiple fields such as criminology, psychology, psychiatry and forensic sciences [Tur11]. Further, it is based on two basic assumptions: (1) behavior is predictable, and (2) people are creatures of habit.

While profiling with the intent of improving chances to catch criminals has probably been done for a long time, one of the very first recorded cases is connected to Jack the Ripper. Jack the Ripper is the name given to an unidentified serial killer who is attributed with the brutal killing of at least five prostitutes in the Whitechapel district of London in the autumn of 1888 [BB17]. At this time, the extent and cruelty of these crimes was unprecedented, which posed new challenges for the investigating police force. The style of these killings led many people to believe that the killer was a trained butcher or slaughterer. In the efforts to identify the murderer, a police surgeon was consulted. This surgeon examined the bodies of the victims and provided a report on his findings. In this report, which has been declared as the first offender profile in history, he asserted that all murders were committed by the same man. However, he opposed the suspicion that this man had any kind of anatomical knowledge, or even any skill regarding the slaughter of livestock [Can94].

This early example of offender profiling did not follow any structured approach. Having a structure in place for this kind of endeavor is however very important in order to ensure objectivity and avoid prejudice or bias. It is mandatory that the methodology is sound and the applied inference rules are transparent to make sure that the results are legally admissible. Even in 1960, when the FBI started to teach courses on profiling, most of the techniques were based on intuition and subjective experience [Tur11]. In the 1980s however, work begun on formalizing efforts and methods, and in 1986 Douglas described what he calls a "criminal-profile-generating process" [DRBH86]. This process consists of six stages, namely the profiling inputs stage, the decision process models stage, the crime assessment stage, the criminal profile stage, the investigation stage, and the apprehension stage.

Since then, offender profiling has continually been developed further. This development resulted in a host of closely related disciplines, such as behavioral profiling, crime scene profiling, criminal personality profiling, psychological profiling, criminal investigative analysis, and investigative psychology. While most of these disciplines share many similarities to the point where they are even used synonymously [Tur11], their individual differences are not of interest within the scope of this thesis.

Summary of offender profiling

Objective	Identify the offender of a committed crime in order to assist in his apprehension
Input	Information about previous offense behaviors, the offender, and the circumstances
Processing	Comparison of previous cases with current case. Similar cases are used to narrow down the list of suspects
Output	Profile that describes key characteristics of the suspect

2.2.2 Crime Prevention Profiling

Also related to law enforcement is the practice of *crime prevention profiling* (sometimes also called *predictive profiling*). As opposed to offender profiling, it is not concerned with identifying perpetrators of already committed crimes, but rather aims at predicting crimes based on previously established profiles of likely perpetrators. Nielsen describes crime prevention profiling as the "recognition of characteristics and behavioral patterns of persons who are likely to threaten security" [Nie05, p. 75]. These characteristics could include anything observable about a person, such as clothing, facial expression, etc. Note that when the assumed race of a person is used in profiling, the term *racial profiling* is usually used instead, which will be discussed in Section 2.2.3.

Simply put, crime prevention profiling aims at predicting crimes or similar offenses, and at initiating measures to stop them from ever being committed. While this may sound like a futuristic plot from a Hollywood movie (e.g., *Minority Report* from 2002), it is much more simple in reality. For example, a security guard at a train station is patrolling the main area and observes passers-by. His job is to provide safety by stepping in when there is a fight or a theft, or other misbehavior by anyone. However, he is not only reacting to offenses when they are happening, but also proactively watching out for suspicious and irregular behavior. As such, he is trying to *predict* when and where an offense might occur. However, he is also *profiling*, because his judgment is based on previous experiences of past offenses. For example, he might know that groups of adolescent boys are more likely to cause trouble than elderly seniors, which is why he is keeping a vigilant eye on the former. This is often sufficient to maintain order and security.

This manual process is increasingly supported through the use of modern technology, most prominently video surveillance. In high security locations, like nuclear power plants or government buildings, video surveillance is commonplace. These video feeds are usually monitored by security staff members, who look for irregularities in all monitored places at once from a central office. This task is repetitive, tiring, and most of the time even boring, which makes it a prime candidate for being automated with computers. This requires first that a computer is able to analyze the content of a video feed in real time, which is the primary focus of the machine vision field [Dav12]. With the ability to *see* a video, a computer then needs to be taught when and how to act. The simplest approach is to configure a set of event-condition-action rules. For example, a rule could read "if a person enters this area at nighttime, raise an alarm". In order to prevent false alarms, such a system needs to be able to differentiate, e. g., between a person and an animal. More sophisticated algorithms have been developed that are even capable of recognizing specific actions, like falling, stealing, or

fighting [HVL08].

Another approach is to design a system that is able to observe a scene and learn autonomously to distinguish between right and wrong. The obvious advantage of this approach is that no laborious setup of rules is required, and adapting to new environments takes less effort. Such a system has, for example, been implemented by Behavioral Recognition Systems, Inc. (BRS Labs) with the fitting name *AISight*. It makes heavy use of machine learning algorithms and is capable of analyzing video content, learning what normal activity looks like, and then raising an alarm when abnormal or irregular activity occurs [27]. Among other places, this system has been deployed in Boston after the 2013 marathon bombings. Authorities hope that this will prevent or at least deter similar attacks in the future and thus, increase security in public spaces. Whether or not this can be a successful approach is still an open question.

Another example that shows the power of prediction with automated surveillance can be found in Japan, where the operators of a train platform have installed a system that can detect through cameras when people are intoxicated and notify attendants if necessary [38]. The detection of drunkenness is based on common behavior of people under the influence of alcohol, such as sleeping on a bench or being motionless for prolonged time. The goal is to prevent accidents of people getting hit by trains, which in the majority of cases happens to drunk people.

Some people even go so far as to claim that the personality of a person can be predicted based on an image of their face. The company *Faception* has developed such a technique under the name *facial personality profiling*. They claim that their "breakthrough computer-vision and machine learning technology analyzes facial images and automatically reveals personalities in real-time" [8]. This goes beyond similarity-based face recognition algorithms that can spot an input face in a video feed. Instead, previously unknown potential offenders can be identified and apprehended before damage can be done. So far, the claims made by the vendor are unsubstantiated and cannot be verified.

Crime prevention profiling is the basis for predictive policing which has been developed at the Los Angeles police department. In predictive policing, an algorithm is fed with crime statistics and other relevant data in order to predict which areas of a city are most likely to see a crime. Patrolling police officers are then sent to these locations to deter any potential miscreants through mere presence. Initial evaluations show that predictive policing has twice the accuracy of usual prediction practices employed by the police [9]. This success has led to the foundation of PredPol [28], a company that focuses on refining the prediction technology and selling it to police departments and other jurisdictions.

It must be noted here that ubiquitous surveillance coupled with automated behavioral analysis unquestionably raises numerous privacy concerns. However, this is a very complex topic which shall not be further discussed here.

Figure 2.2: Satirical depiction of "hard" racial profiling. Source: Family Guy, Fox Broadcasting Company.

Summary of crime prevention profiling

Objective	Predict time and place of a crime before it can be committed
Input	Real-time information about a specific environment; usually in visual form (e. g., a video feed)
Processing	Comparison of input with rules or previously observed events
Output	Alert of suspicious behavior, apprehension of suspect, dispatch of a patrol to designated area

2.2.3 Racial Profiling

Racial profiling describes the practice to suspect a person based on their race and/or skin color. Unlike offender or crime prevention profiling, it is not so much an established technique that has been developed deliberately, but rather a label that has been assigned to the behavior of specific individuals or groups. The most prominent use of the term racial profiling occurs in the context of law enforcement, especially when police officers stop and interrogate colored people [WT02].

In order to pave the way for a more rational discussion about this subject, Heather Mac Donald suggests a distinction between *hard* and *soft* racial profiling [6]. Hard profiling takes place when race is the *only* considered factor when looking out for potential criminals. With soft racial profiling on the other hand, race is only one factor out of many that is used to assess an individual and his level of suspiciousness. In this sense, soft racial profiling is related to crime prevention profiling as described before. Figure 2.2 shows a satirical example of hard racial profiling as seen in the adult animated sitcom *Family Guy*. In this picture, the driver of a car is stopped and his skin color is assessed with the use of a chart. The chart shows six different shades of skin color, with the lighter three ones labeled as "okay", and the darker three ones labeled as "not okay", indicating that it is suspicious to be a member of a race with dark skin color.

Racial profiling is a highly debated topic, because discrimination based on the race of a person is considered racism, which is illegal in most countries. Therefore, attempts have been made to ban the practice altogether, for example in the US by president Bush in 2003 [16]. However, it is not trivial to enforce such a ruling, because it is often impossible to provide

evidence that race was a determining factor in an event or action. Furthermore, it needs to be said that the race of a person is just another item in the list of demographic attributes (next to age or gender) that can be gathered about an individual. Other people even argue that including the ethnicity of a person in an offender profile is a successful technique for crime-fighting that should be continued [6].

Summary of racial profiling

Objective	Not explicitly defined. Usually to prevent crimes or misdemeanor
Input	Observation of people's skin color or ethnicity
Processing	Comparison of input with previous experiences
Output	Decision to take an action against a person of a specific race

2.2.4 Customer Profiling

Marketing professionals are always looking for new and improved ways to sell their products and satisfy the needs of their (existing and potential) customers. In order to do so, frequent analyses are carried out regarding a multitude of influencing factors. One such analysis is called customer profiling, which is concerned with the creation of customer profiles and their exploitation to increase sales. A customer profile acts as a model of the customer and his behavior, and serves as a basis upon which a market practitioner can make his decisions [SSTW01]. Depending on the application context, a customer profile consists of different attributes. These usually include demographic attributes (e.g., age, income, gender, education, occupation, etc.) and psychographic attributes (e.g., hobbies, concerns, political beliefs). A customer profile can either be used to portray a single real-world customer of the company, or act as a model for a group or segment of like-minded customers which share similar attributes. The former is usually employed when the number of individual customers is low and a descriptive analysis is desired, while the latter is more useful if there are numerous customers whose behavior needs to be predicted.

As an example, consider a local supermarket in a small town. The analysis of sales data reveals a pattern of customer behavior throughout the day: during the early hours, mainly housewives and -men come in, and in the evening, more students are shopping. Furthermore, sales analysis reveals that the housewives and -men tend to buy more expensive brand products, whereas students go for the cheaper no-name products. This information constitutes the customer profiles, which can be leveraged by management to tailor the shopping experience to the targeted customer segment based on the time of day. For example, the promotion aisle with special offers can be stocked with brand products during the day and no-name products in the evening. This strategy is then likely to support the goal of increasing sales.

This example assumes that demographic attributes of the customers can easily be observed, which is usually not the case in a classic retail setting where people want to get their shopping done and not be bothered by questions or surveys. However, with the rise of the Internet in general and e-commerce in particular, more and more purchases move away from brick-and-mortar stores and into online channels. Marketing researchers have quickly found out that these online channels offer a wealth of information about their users which can easily be extracted. The information sources include server logs, session cookies, and related mechanisms and technologies, which are commonly referred to as Web analytics [PSF04].

These sources allow insights into various characteristics and behaviors of a user, such as which Web sites he has previously visited and for how long, or which hyperlinks and ads he has clicked on. As these kinds of information are a natural byproduct of Web site operation, no additional effort needs to be made to get this raw data [WBW02]. In order to transform these data into useful customer profiles, additional processing and analysis needs to be carried out. Ideally, this includes the integration of other data sources (such as social media accounts) to augment and complete the profiles with demographic and psychographic attributes. These online customer profiles can then be used to, for example, provide customized advertising specifically tailored towards the interests of a user.

Customer profiling is usually targeted at the end consumer of a product or service, which means it is used in B2C (*business to consumer*) relations. However, it is also possible to perform profiling of customers in a B2B (*business to business*) setting where the customers are other companies. Hence, this practice is sometimes called company profiling and includes different attributes, but similar methods.

Related to customer profiling is the idea of a recommender system [RV97][LRU14, p. 307]. The goal of a recommender is to provide a user with recommendations of items he might potentially be interested in. These items can be products, books, movies, music, or any other object with distinguishable and comparable attributes. One type of recommender system learns the preferences of its users over time by monitoring their behavior and keeping track of their assessment of encountered items (such as ratings or reviews). This information is then used to either find similar items to those a user liked, and recommend them, or find similar users, and recommend items they have liked in the past. So even though the word *profiling* is not explicitly used in the discussion surrounding recommender systems, there is a strong similarity between the methodology of customer profiling and recommender systems.

Summary of customer profiling

Objective	Help marketing decision making by gathering customer data and analyzing customer behavior
Input	Various sources of customer information, e.g., social media, sales records, or clickstreams
Processing	Aggregation and summarization of input sources
Output	Customer profiles

2.2.5 DNA Profiling

DNA profiling is concerned with creating genetic profiles to uniquely and unambiguously identify individuals. It is sometimes also referred to as DNA fingerprinting because it shares many similarities with the usage of fingerprints in criminal investigations. The roots of DNA profiling go back to the findings of the geneticist Sir Alec Jeffreys. In 1984, Jeffreys discovered by pure chance that DNA contains extractable patterns that are unique to an individual, with the exception of identical twins. Although he could not foresee the wide applicability of his findings, he described his findings on DNA fingerprints as being able to "provide an individual-specific DNA 'fingerprint' of general use in human genetic analysis" [JWT85]. After more refining of this technology, Jeffreys developed a test that could be used to assess whether two people shared enough genetic characteristics that they can be declared

as blood-related. It did not take long until the true potential of such a test was discovered: In immigration cases, where lawyers fighting deportation claimed maternity or paternity, a legally valid proof could now be obtained [17]. Modern paternity tests, which are much cheaper and quicker, are based on the same principles discovered back then.

Another application area for DNA profiling is, once again, law enforcement. Due to the fact that perpetrators often leave traces of their DNA behind at crime scenes and that DNA can be unambiguously attributed to a specific person, many cases could be solved with the help of DNA profiling. In fact, many TV shows and movies are built around the practice of finding single hairs or droplets of blood to assist crime scene investigation and police work. Once a source of DNA has been found, it needs to be processed into a DNA profile in a laboratory. This profile can then be compared against a database that maps DNA profiles to persons. These databases, aptly named DNA databases, are highly controversial because a person, once recorded, cannot demand the deletion of their DNA profile. This is technically a breach of data privacy, purely based on the suspicion that this person might commit a crime in the future. The trade-off between civil liberties on one side, and state security on the other side, is a recurring topic in public debate.

Summary of DNA profiling

Objective	Uniquely identify individuals to settle paternity/maternity disputes, or find perpetrators of a crime
Input	DNA source of person in question; database with previously recorded profiles
Processing	Extraction of unique characteristics from the source material; compilation into profiles; comparison
Output	Degree of similarity between two or more profiles; probability of identity or paternity/maternity

2.2.6 Software Profiling

Software profiling (also called program profiling) is used in software development to assess and benchmark a given piece of source code. It is different from software testing in the way that profiling is about identifying bottlenecks in a program, whereas testing is about finding bugs and other unwanted behavior. A developer who wishes to profile his code usually does so with the use of a (code) profiler. A profiler is a software component which is commonly integrated directly into the development environment, although stand-alone profilers also exist. Profilers operate in one of two different fashions, *offline* or *online* [DB00]. Offline profiling means that the code to be profiled is still under development and not part of a running system yet. Execution takes place in a dedicated run of the program whose only purpose is the collection of various characteristics of the program, such as memory consumption, runtimes of functions, usage of instructions, etc. The resulting profile is a summary of the program, which is then analyzed by a software developer to spot anomalies or bottlenecks, and subsequently improve and optimize the input source code. Online profiling on the other hand is applied in dynamic compilation systems, and the collection and consumption of profile information occurs within the same run [DB00]. Thus, this approach requires a high degree of efficiency, i. e., little overhead caused by the profiling mechanism. It allows monitoring and benchmarking of a system while it is running in production, which can give more realistic results compared to

the artificial setting in offline profiling.

Orthogonal to the distinction between online and offline modes is a classification based on *how* a profiler gathers its data. The most commonly encountered methods are instrumentation, event-based, and statistical, which will be discussed in the following paragraphs.

Instrumentation is the oldest technique of software profiling. It consists of inserting measurement code into a program, executing it, and measuring the output [BL94]. Generally, this insertion adds considerable overhead to a program's runtime, which might cause problems in an online setting. Estimates put this overhead anywhere between 3% and 40% [Wu03]. The degree of incurred overhead heavily depends on the placement of the measurement code which in turn depends on the required granularity of the measured data.

Statistical profilers rely on sampling to periodically poll the status of a program (e. g., the call stack or the memory allocation) while it is running. This approach is numerically less accurate, but causes less overhead. For example, the Digital Continuous Profiling Infrastructure described by Anderson et al. is reported to only have 1-3% overhead while being around 10% off on average [ABD+97]. Thus, this approach is preferred in scenarios where latency is more important than accuracy.

Event-based profilers use native interfaces provided by the programming language to hook into specific events. These events are triggered by the program during runtime and include cases such as function calls, memory allocation and thread changes. One use case for event-based profiling is refactoring of graphical user interfaces, as described by Nagarajan and Memon [NM03].

One key difference that distinguishes software profiling from the other types of profiling described before is the fact that usually no profile comparison takes place. The created profiles are either consumed immediately or analyzed after the run. This implies that storing of software profiles is not needed, because their purpose can be directly and immediately achieved. Furthermore, a comparison of software profiles from different programs makes little sense, because every source code is different and there are hardly any code-specific profile characteristics that can be contrasted. For example, the frequency and duration of function calls is individual to every program, because every program defines its own unique functions. In cases where a comparison of different programs is needed, like a comparison of different implementations of the same algorithm, software profiling is not the correct technique to be applied. Rather, benchmarking should be used, which shares some similarities with profiling, but ultimately uses its own techniques, methods and tools.

Summary of software profiling

Objective	Analyze source code to identify optimization potential
Input	Source code of a program
Processing	Monitored execution through a profiler
Output	Software profile that gives insight into where in the code a program spends its time and resources

2.2.7 Data Profiling

Data profiling is the practice of extracting metadata to learn about the characteristics of a given dataset. In other words, Abedjan et al. have defined it as "the set of activities and processes to determine the metadata about a given dataset" [AGN15]. In this context, metadata is any piece of information that describes or quantifies a characteristic of the underlying dataset, e. g., the number of rows or the distribution of values. The extraction of such metadata is usually carried out with specialized tools, but can also be done with targeted queries or even manual counting. After extraction, the resulting metadata is compiled and stored in the *profile* of the input data. A profile can then be displayed in tabular or graphical form and be used in a wide range of use cases, ranging from simple data validations over quality assessments to complex data integration scenarios.

Data profiling is not new. In fact, the use of statistical profiles for query optimization has already been described almost thirty years ago [MCS88]. Since then, many techniques and algorithms have been developed, although data profiling was never really considered to be a research area in its own right [Nau13]. As data profiling is a major focus of this thesis, this section only provided a high-level view of it. An in-depth explanation of the current state-of-the-art in data profiling and all its constituent components will be given in Chapter 4.

For the sake of completeness, it should also be mentioned that there is a similarly named but functionally different discipline called *database profiling*. It is not concerned with the data itself, but rather with the performance of a database in execution [MPdS+08]. In this sense, database profiling is similar to software profiling, because here the goal is also to find bottlenecks and performance hogs by analyzing the posed instructions and queries. The derived insights are then used to optimize query design and ultimately guide overall software architecture design [MPdS+08]. Database profiling is mainly concerned with the operations of a database and how they are used, such as joins, storage, and memory load, whereas data profiling focuses on the intrinsic characteristics of the data itself, independent of the system that is used to handle it.

Summary of data profiling

Objective	Extract metadata to aggregate, characterize and summarize data
Input	Raw data, datasets, or collections of data
Processing	Profiling algorithms and tools
Output	Data profile consisting of metadata and charts

2.3 A Model for Profiling Activities

Now that the most common application areas for profiling have been described, a generic profiling model can be developed to synthesize commonalities. This model can be used to explain and structure existing profiling activities as well as classify new ones. Furthermore, the model helps diving deeper into the intricacies of data profiling throughout the remainder of this thesis. Figure 2.3 shows the model and the rest of this section explains its components as well as the reasoning behind them.

The first observation is that every profiling activity is focused on an object that needs to be characterized or summarized. This makes sense because a profile is always a profile of

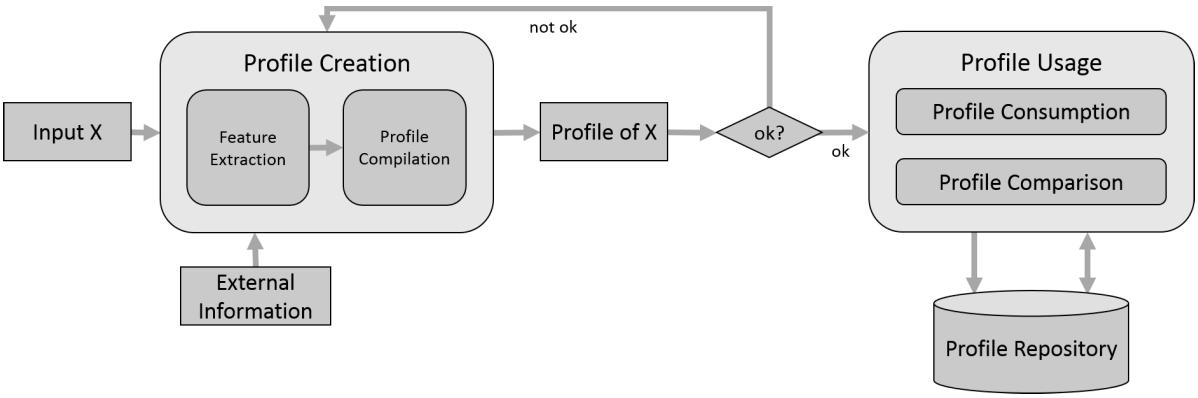

Figure 2.3: Generic model for profiling activities.

something and that something was used to create the profile in the first place. This to-be-profiled object shall henceforth be called *input*. For example, the input could be a DNA sample in DNA profiling, or an image of a person in offender profiling. Next to the input object itself, further information surrounding it could be available. This information is not part of the input object itself, and thus shall be referred to as *external information*. This external information could, e. g., consist of the context from which an input object originated. For example, the location from which a DNA sample has been taken, or the name of the witness describing a criminal would be considered external information. The consideration of these additional information sources can enhance and guide profile creation by narrowing down the set of possible options. If for example a DNA sample has been taken from an area with restricted access, it is likely that it belongs to a person who had access to that area. After the input and external information have been declared, the first activity can begin.

2.3.1 Profile Creation

The input and external information are both fed into profile creation which consists of two consecutive steps, *feature extraction* and *profile compilation*. For the extraction step it is necessary that it was previously defined *which* features are of relevance for the profile. For example, during customer profiling, it needs to be clearly stated which attributes of the customers should be recorded, like age or annual income. For every feature, a procedure is required that describes *how* the extraction is performed and which source it is derived from. In customer profiling, some features (like age or place of residence) could be extracted using surveys or direct interviews, while other features (like income or social status) are better sourced from a third-party data provider, e. g. the Socio-Economic Panel. Once all required data about features has been collected from the various sources, it needs to be put into a structured form to facilitate its later usage. This structuring step shall be called *compilation*, because it aggregates all the individual profile pieces and assembles them into one cohesive profile. The complexity of this compilation depends on the requirements that resulting profile must meet. For example, in customer profiling it is most important that a profile can easily be communicated to the decision makers. Thus, it is desirable that a customer profile contains visual elements, is concise, and not too convoluted with numbers and unnecessary details. In

offender profiling, it is important to focus a profile on the appearances of a suspect so that he can be found and identified in a crowd (think of a wanted poster in the Wild West). Vastly different requirements must be fulfilled by a DNA profile: here, the purpose is not to be visually digested by a human, but rather to be readable and comparable by a computer. Accordingly, the compilation step consists of transforming the raw feature data into a machine-readable format that can be stored and persisted in an appropriate database.

The output of profile creation is the profile of the input object. Now, it is interesting to think about how this relationship between input objects and profiles can be quantified. A profile is always derived from the input object that was used for its creation, so there is always a precise assignment from a profile to an input object. Easier put, every profile belongs to *one* input object. This also holds true in cases where two input objects are so similar that their resulting profiles are identical, because each profile has its own identity. If the input object was a collection of elements (e. g., a set of tables in data profiling), then the profile is assigned to the whole collection as one object, and not to its individual, constituent components. At this point, it should be pointed out that it makes little sense to create a profile of a collection just by merging the profiles of the individual elements from the collection, as Aristotle's famous saying "the whole is more than the sum of its parts" applies.

Looking at the other direction of the relationship between input objects and profiles, the cardinality is less obvious. One could argue that for every input object exactly one profile should exist. However, this perspective is not very useful in this generic view as it neglects many circumstances. First, the context influences the profile creation. A piece of software (input) performs differently depending on the hardware it is run on (the context). Secondly, the profile creation procedure may be subject to change. When the selected features or steps for extraction are modified, it is inevitable that the resulting profiles are different. As such it is concluded that any input object can have one or more profiles. In some cases, it makes sense to limit this cardinality to exactly one. For example, a human being should have exactly one DNA profile in order to allow for unambiguous identification. Implicitly this means that the context of the DNA sample and the extraction method should have no influence on the resulting profile.

After the profile has been created, it is checked whether it is good enough for the task at hand, i. e., if all requirements and conditions are met. This inspection step has two possible, mutually exclusive results, *ok* and *not ok*. In the case of *not ok*, the profile goes back to the creation phase where it is refined. For example, it could be discovered that a DNA profile does not contain the correct features. During the next iteration, a different configuration for the feature extraction is chosen to remedy this issue. Similarly, the fault of a profile could also stem from the compilation step. If for example a customer profile is not suitable for presentation, then it should be revised with more appropriate compilation procedures, such as better visualization. Overall, the profile inspection introduces a feedback loop that ensures that all output profiles are of sufficient quality and meet the laid-out standards.

2.3.2 Profile Usage

Profiles that pass the inspection step are passed onto the profile usage phase. In general there are two distinct types of profile usage: profile *consumption* and profile *comparison*. Profile

consumption means that the profile is consumed by a process that directly transforms it into some kind of value or insight. For example, a software profile is consumed by a software developer through analysis and assessment. The developer learns characteristics of the profiled source code and uses these insights to optimize or refactor it. After that, the software profile is no longer valid because the input object has changed. Generally speaking, a profile that has been consumed has fulfilled its purpose and can thus be discarded. Specifically, it does not need to be stored.

Comparison of profiles works in a different way. First of all, a *repository* is necessary to store all previously created profiles. This requires that those profiles have been compiled in a way that makes them suitable for storage in a repository. Furthermore, the format and structure of all stored profiles need to be similar enough to allow meaningful comparison. Then, as soon as a new profile comes in, a system can automatically perform the comparison. Consider for example a DNA database and an unidentified DNA profile. The system can search for a match of the profile in the database which, if successful, allows identification. If no full match is found, i. e., not all profile components are equivalent, even a partial match can be helpful as it might indicate a blood relation (sibling, parent, or child) between the two profile holders. For this and related use cases, a comparison algorithm should also output some kind of confidence measure that expresses the quality of the returned matches. This is of great relevance in crime prevention profiling where the repository stores abnormal reportable behavior and the input is the video feed of a surveillance camera. In this setting, the input is constantly matched against the repository and evaluated into a suspicion score. Once this score exceeds a threshold value, an alarm is raised. Careful consideration should be placed upon calibrating this threshold value in order to minimize false alarms and maximize the chance of rightfully alerting abnormal behavior.

2.4 From Description to Prescription: A Classification

To further structure the various profiling activities, it is useful to look more closely at their respective purpose and considered time horizon. A common classification for these characteristics distinguishes between three main perspectives: descriptive, predictive, and prescriptive [14]. This section first introduces these perspectives and their corresponding disciplines in data analytics. These insights are then transferred to profiling, where they are used to classify the previously described profiling application areas.

The *descriptive* perspective is about gathering and presenting facts about events that have taken place before. Thus, it is retrospective. On the other hand, the *predictive* and *prescriptive* perspectives are prospective in the sense that they regard the future and the events that are most likely to take place. To this end, statistical models are used to learn from the past and extrapolate this experience into the future. This is done under the assumption that the past will repeat itself to at least some degree, i. e., the same input will lead to similar output. The difference between the *predictive* and *prescriptive* perspectives is that the former assumes the role of a passive observer and only predicts what will happen, whereas the latter also considers possible actions of an acting entity and their effects on the predicted future.

These three perspectives have corresponding techniques in the fields of data analytics and

data mining. Descriptive analytics is the most basic variant. The underlying question "what has happened?" is answered through charts, reports and dashboards that show aggregate statistics about past events. These results are used, e. g., by managers to make informed business decisions. Due to its low complexity and wide applicability, descriptive analytics are at the core of virtually any business intelligence project. In addition to the informative value itself, descriptive analytics can also form the basis for more complex and sophisticated techniques.

Predictive analytics goes a step further and addresses the question "what will most likely happen?". To this end, past performances are analyzed to build a model that is able to extrapolate into the future and to predict events and occurrences that have not yet taken place. A prominent use case for predictive analytics is *churn prediction* (also called *churn modeling* [Sie16]) which aims at predicting which customers of a business are most likely to churn, i. e., cancel their subscription or contract. If a company is able to successfully identify the customers that are most likely to leave before they actually do so, it can initiate measures to retain these customers, e. g., offer special promotions.

Finally, prescriptive analytics subsumes predictive analytics and extends it in regard with possible actions that can be taken to affect the predicted future. As such, it poses and aims to answer the question "how can we change what will happen?". This requires further input to the model about actions that have previously been taken, what their results were, and which actions can still be applied in the future. Due to this increase in complexity, much more sophisticated techniques and algorithms are required, which are usually taken from fields such as machine learning, operations research or management science [BK14]. Prescriptive analytics is especially valuable in scenarios where the actions of the decision maker have a strong impact on the surrounding environment or market. For example, a large investor in a stock market has the potential to heavily influence the stock prices with his decisions to buy or sell. Thus, including his own actions can increase the accuracy of the model used to predict future prices.

The three described perspectives are used to classify the profiling activities with the result being shown in Figure 2.4.

Offender profiling is first and foremost about portraying a criminal or offender. It is focused on the appearance and superficial characteristics of individual persons and how this information can be provided to other people to raise the chances somebody will recognize the person in question. Thus, its main purpose is description. However, it can also be argued that a prediction pertaining to the real identity of a perpetrator is made, and that future crimes from that person need to be prevented. Following this line of thought, there is no clear line to be drawn between offender profiling and crime prevention profiling, as indicated by the merging of both into one bar in Figure 2.4. Note that a prescriptive purpose can also be found in some use cases of crime prevention profiling, such as a recommendation to send out a police patrol to a certain area at a specific time. Racial profiling on the other hand, although not a real discipline in its own right, deals with a purely descriptive perspective, such as the color of a person's skin or other factors of race and ethnicity.

Customer profiling spans the whole spectrum. This starts with profiles about individual customers that usually contain a list of characteristics as well as a history of past interactions, e. g., sales. These profiles serve a descriptive purpose. Based on these, prediction models can

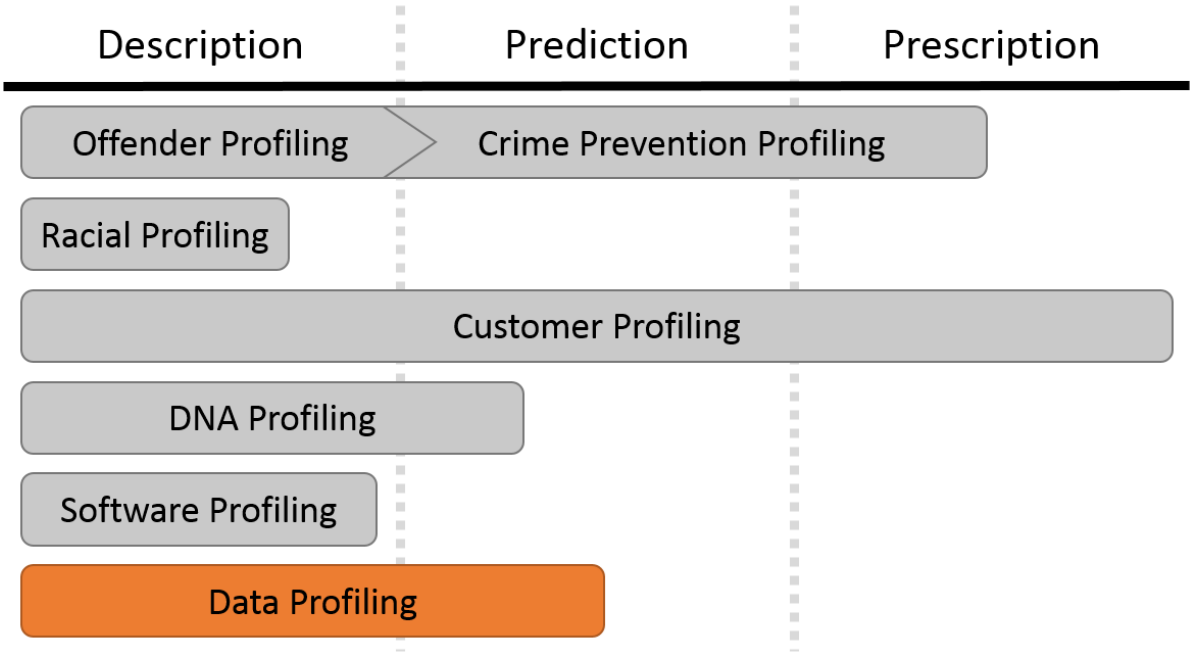

Figure 2.4: Classification of profiling activities according to main purpose.

be set up in order to estimate how well promotions or other marketing activities are perceived. In this sense, customer profiling can also be prescriptive as the potential actions are also part of the model. The goal of DNA profiling is to obtain a unique genetic fingerprint of an individual, which is a descriptive purpose. However, there also exist procedures that have a more predictive nature. For example, the DNA of a person can be checked to predict whether he or she is susceptible to a hereditary diseases or has any genetic disorder. In software profiling, statistics about the runtime of a given piece of code are generated and analyzed. To obtain this information, the code is executed, resulting in empirical results that are purely descriptive. It should be noted that there also exist approaches in which this information is predicted [HXHLB14]. However, these approaches are not called profiling, and thus, are not considered here. Data profiling is primarily about describing a given dataset through usage of metadata and visualizations. In the CloudHost case, these descriptions are used to gain a better understanding of the available data about its customers. Depending on the context however, a data profile can also be used to make predictions. For example, future data growth or schema changes can be estimated from a data profile. For CloudHost, this can enable them to better plan and provision their resources (e. g., their available disk space per customer), or even to directly support business decisions (e. g., which customers to prioritize based on their usage of CloudHost services).

In conclusion, profiling is an activity that is used in many diverse application areas with varying objectives, processing techniques, representation types, purposes and time horizons. After this general introduction into various application areas of profiling, the remainder of this thesis will focus on data profiling specifically.

3 Problem Identification and Characterization

More and more data is generated every day. There are many reasons for this development, such as the massive proliferation of mobile smart devices that allow virtually anybody to create data (pictures or videos, geographic data, social media content, etc.) at any time. At CloudHost, new software products and systems are introduced continually, and each product generates data during its operation. As each piece of data is potentially useful, nothing is discarded right away, and all data is stored.

On the other hand, the storage capacities and processing speeds of the systems that deal with this influx of data are continually improving. This never-ending circle of increasing dimensions (volume, variety, velocity [Lan01]; also referred to as Big Data [CML14]) has usually one limiting factor in it: the human mind. Our cognitive capacities have a natural upper bound that limits how much information we can take in, process, act upon, and remember. Even though these cognitive capacities have been increased over time through evolution and education, this growth is orders of magnitudes slower than the corresponding increase in computing power observed over the last decades. Furthermore, it is reasonable to expect that this gap will only get larger in the foreseeable future.

The CloudHost company is also facing this issue: As they expand their business and grow their number of clients, they are confronted with increasingly heterogeneous requirements they need to meet and data sources they need to integrate and manage. For example, a new client might want to run a novel piece of software on the CloudHost platform. Ensuring a smooth hosting of this software requires a monitoring of its logging data, which in turn requires an understanding of what this data means.

Overall, the Big Data phenomenon leads to an increasing frequency of problems where individuals are confronted with datasets that are too complex to make sense of. These problems are particularly severe when little or nothing is known beforehand about that dataset. In this chapter, the intricacies of this problem and its characteristics will be investigated. To this end, Section 3.1 focuses on the data as the input to the problem. Section 3.2 describes typical tasks and objectives in which the problem occurs, while Section 3.3 highlights information overload as both its cause and effect. The desired outcome is data comprehension and is explained in Section 3.4. Section 3.5 covers a characterization of the users that are facing the described challenges, and addresses the question what changes when that user is a group of people instead of a single individual. Finally, Section 3.6 shows how the described problem is usually approached, and how that could be improved.

3.1 Unfamiliar Datasets

The term *dataset* describes a cohesive collection of data values, which can be structured in any arbitrary way. Examples of common datasets are spreadsheets, databases, documents, and tables. With increasing size and complexity, the efficient analysis and processing of datasets is becoming more and more difficult. This issue is exacerbated in cases where the dataset is new to the person dealing with it, which shall be denoted as being *unfamiliar*. Unfamiliar datasets are encountered frequently in any data-driven environment. For example, whenever CloudHost deploys a new software, they are likely to be confronted with data that they did not deal with before. In such a case, they do not know anything about the dataset, its content, or its structure. This unfamiliarity prevents an individual from being able to efficiently carry out any task with regard to that dataset, like analysis or monitoring.

Similarly, when a company acquires or buys a new dataset to augment its internal data sources, the provided information about it may be incomplete, e. g., the documentation is outdated or missing, or the quality of the data is uncertain. Even if this information were to be included, it needs to be confirmed before the dataset is used in a productive setting. Thus, one should always err on the side of caution, and verify that any claims that are made about a dataset hold true. For example, if an integrity constraint is documented, it must be ensured that it has been properly enforced. If an attribute is said to be 100% complete, it is easier to check it than to blindly trust the information and fix errors that appear later when a violation occurs.

In general, datasets can be highly dynamic and may change over time, which generates the recurring issue of having to deal with unfamiliar datasets. The unfamiliarity state of a dataset is a naturally subjective characteristic that is dependent on the person looking at it. This means that one dataset can be unfamiliar to one person while at the same time being familiar to another person. Additionally, one's capability to read and make sense of data plays a crucial role in the assessment of the issue at hand. This capability is commonly referred to as *data literacy* [MG13].

The (un-)familiarity of a dataset to a person can be thought of as a spectrum ranging from completely unfamiliar to completely familiar. On this spectrum, one can move towards familiarity by learning about the dataset by inspecting and profiling it, and move towards unfamiliarity over time by forgetting about it or when the dataset is changed and updated. Most cases are somewhere in between the endpoints of this spectrum, as on the one side there is usually at least something to start with, like an assumption or informal information (i. e., the dataset is not completely unfamiliar). For example, if there is any similarity of a new dataset to one that has been encountered before, an experienced CloudHost employee might use his knowledge to grasp the new dataset quicker. This preconception is strengthened through the observation that best practices and database design principles are often enforced, which leads to similarly structured datasets that are not completely unfamiliar to experienced people. Looking at the other side of the spectrum, there is almost always something more that can be learned about the dataset (i. e., it is not completely familiar). For example, even if a senior CloudHost employee has curated a specific dataset for the last ten years and claims to know it inside out, there is a high chance that there remains a piece of information that he is not aware of. This could be a hidden correlation between attributes, the value of a data

quality metric, or something else entirely. Now, it is rarely the goal to reach this far end of the spectrum and to learn everything there is about a given dataset. Instead, a specific level of knowledge is usually required to be able to work effectively with that dataset and carry out the task that has been set. Once this level has been reached, i. e., sufficient knowledge has been gathered about the dataset, work can commence or continue. Due to the significance of this level, it shall be given the distinctive term *data comprehension*, which is further detailed in Section 3.4. Achieving data comprehension is a non-trivial process that is naturally dependent on the size and complexity of the dataset. Small datasets with low complexity (e. g., the list of all CloudHost employees) can be self-explanatory and understandable upon first glance. This and other ad-hoc approaches are further laid out in Section 3.6. These approaches however fail when the size and complexity increases beyond certain thresholds (e. g., more data than can fit on one screen, non-descriptive attribute names, etc.). Then, a more sophisticated data profiling strategy is necessary to turn an unfamiliar dataset into a familiar one and achieve the desired data comprehension. The remaining chapters of this thesis will highlight some of these strategies as they are described in the literature, as well as introduce novel approaches to data profiling.

A dataset can be structured in any arbitrary way, as explained earlier. The remainder of this section discusses the most relevant ways of structuring data and their characteristics.

3.1.1 Relational Data Model

The most prominent data model is the relational data model, which was invented by Codd in 1970 [Cod70]. Ever since then, it has been continually advanced and optimized and became the most dominant data model for a wide range of applications and systems. It is conceptually rooted in relational algebra (hence the name) [Vos91, p. 97] and set theory in order to provide an abstraction layer between the internal organization of data in a machine and the user interface for querying and manipulating the data. This interface is provided by means of a declarative query language that allows a user to specify what data he wants without worrying about how the data is retrieved. The most common query language is the structured query language (SQL), which has been considered one of the main reasons why relational databases became so successful [EN15, p. 207], because it established a standard that allowed users to easily migrate their data between application and database systems from different vendors.

Data in a relational structure can be represented by using tables. For example, Figure 3.1 shows how CloudHost stores information about their employees and associated departments using the relational model. The $\perp$ symbol is used to denote a null value.

One key characteristic of relational databases is that they require the definition of a static schema before any data can be stored. On the one hand, this ensures a consistent structure that every data instance must obey, but on the other hand, it restricts the flexibility of the database with regard to changing requirements. This rigidity has led to the development of other, less strict data models that do not require a schema.

employee				
id	**name**	**age**	**hours**	**dep**
1	John Smith	38	39.75	1
2	Sun Li	42	39.75	1
3	Jane Miller	⊥	19.875	1
4		45	40	2

department	
id	**name**
1	HR
2	Sales

Figure 3.1: Example excerpt of two CloudHost tables.

3.1.2 Key-Value Data Model

The most basic schema-less data model is the key-value data model, in which data is stored in pairs of keys and associated values. The keys are used to identify the values and thus need to be unique within the same collection. In this way, key-value pairs are very similar to associative arrays and dictionaries as known from modern programming languages.

This model only supports basic operations to push and retrieve values to and from specified keys. Thus, it is not possible to look up data according to specific criteria or perform searches of any kind. This limitation of functionality makes the usage very simple. A user can insert new data by supplying it as a key-value pair. For data retrieval, the user can pass a key and receive the associated value, if it exists. The storage engine keeps no track of what is stored in the actual values, which means that there is no query language that allows a search based on any specific criteria or value properties. Due to this simplistic design, key-value stores can be implemented in very efficient ways and typically achieve comparatively fast speeds, both in terms of latency and throughput. This makes key-value stores an attractive choice for use cases in which performance is crucial.

The most prominent use case for key-value stores is caching, e. g., in a web application. Here, the key-value store is not used to persist data, but rather to allow more efficient access to data that resides in another database. For example, *Memcached* [7] provides the specific functionality to cache key-value structured data in memory to speed up web requests and database queries. This can help in scenarios where the database is the bottleneck, either because the data volume is very large or the data is accessed very frequently.

Many modern implementations extend these basic functionalities. For example, *Redis* [31], a widely popular key-value store, offers a number of different data types. Numbers can be stored and interpreted as such, which can be used for counters or other purposes. Incrementing and decrementing numbers are implemented as atomic operations, which helps to alleviate concurrency problems in a distributed setting.

Figure 3.2 shows how the same CloudHost employee data from before might be stored in a key-value store. Note that this is just one possible way to structure the data, and many other ways exist. For example, each employee could be stored as one single key-value pair, with the key representing the id, and the value being a concatenation of attribute values.

Key	Value
emp:1:name	John Smith
emp:1:age	38
emp:1:dep	HR
emp:1:hours	39.75
emp:2:name	Sun Li
emp:2:age	42
emp:2:dep	HR
emp:2:hours	39.75

Key	Value
emp:3:name	Jane Miller
emp:3:dep	HR
emp:3:hours	19.875
emp:4:name	
emp:4:age	45
emp:4:dep	Sales
emp:4:hours	40

Figure 3.2: CloudHost employee example as key-value pairs.

```
{ "employees": [
  { "id": 1, "name": "John Smith", "age": 38, "dep": "HR", "hours": 39.75},
  { "id": 2, "name": "Sun Li", "age": 42, "dep": "HR", "hours": 39.75},
  { "id": 3, "name": "Jane Miller", "dep": "HR", "hours": 19.875 },
  { "id": 4, "name": "", "age": 45, "dep": "Sales", "hours": 40 }
] }
```

Figure 3.3: CloudHost employee example in JSON.

3.1.3 Document-Oriented Data Model

A subclass of the key-value data model is the document-oriented data model, which treats the value part of a tuple as a *document* [EN15, p. 425]. A document in this context is a data structure that encodes data in accordance with a document format, the most popular being XML, YAML, JSON, and BSON. These formats allow the database system to make use of the data content to offer a richer user experience that supports complex queries.

In Figure 3.3 it is demonstrated how the CloudHost employee data might be stored using a JSON syntax.

3.1.4 Graph-Oriented Data Model

The graph data model is based on the mathematical specification of a graph, which is a set of nodes connected by edges [Die17, p. 2][EN15, p. 903]. Data is stored as labels or descriptors of nodes, and edges are used to denote relationships between data instances. This data model is particularly well suited for associative data in which traversing along the edges is a common scenario, e. g., a social network. Internally, many graph databases make use of a key-value data model to physically store their data.

An overview of different approaches for graph database modeling and associated data structures and query languages is given by Angles and Gutierrez [AG08].

The CloudHost employee example interpreted as a graph is visualized in Figure 3.4.

3.2 Typical Tasks and Objectives

This section gives a broad overview of typical tasks that arise in a data-centric environment. Data-centric in this context means that each task revolves around a dataset as input, and processing and producing a result from it is the goal. All tasks have in common that a certain degree of data comprehension is necessary before it can be carried out.

Data Loading One of the core requirements with respect to data is the ability to *load* it into a system or database [EN15, p. 45]. This task, which is also referred to as *data ingestion* [SS13], establishes the basis for many further steps by enabling access to the data. The complexity of the data loading task is largely dependent on its input. For example, loading a SQL dump into a SQL database is just a matter of executing a single command that points to the respective file. As the SQL dump contains all necessary information about the data, such as data types, schema, constraints, etc., no further action is required. This is different for datasets that do not bring these types of additional information with them. A flat comma-separated values (CSV) file for example only contains raw data values with no further context. When such a file needs to be loaded into a database, several challenges arise, such as determination of data types, identification of primary keys (or candidates thereof), and establishment of reasonable integrity constraints.

Data Integration In many scenarios, multiple data sources need to be combined into one logical or physical source. This challenge is well researched under the term *data integration* [Hal01, LM02], which has gained a lot of significance in recent years due to the increasing demand to connect previously isolated datasets. A data integration system provides the user with a single uniform interface to which he can pose his query. The system can then autonomously unfold that query, retrieve the relevant parts from the individual sources, join them into a result and present it to the user. One of the main characteristics of a successful data integration system is that the user is not even aware he is using one, because the source selection and query processing is handled

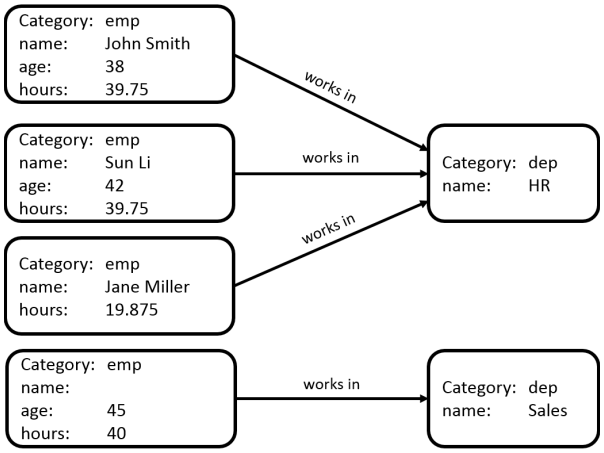

Figure 3.4: CloudHost employee example as a graph.

transparently in the background. This allows the addition and removal of data sources on the fly without interrupting operations or breaking queries.

Data Migration Moving data from one location to another is called *data migration* [HHK+01]. This includes movement in a physical sense regarding the hardware (e. g., from one hard drive or datacenter to another) and movement in a logical sense regarding the data management system (e. g., from one DBMS to another). There is a number of reasons for data migration, such as load balancing, system expansion, system exchange, or failure recovery [LAW02]. The simplest form of data migration is copying, where the data is moved as it is from a source to a functionally equivalent destination. It gets more difficult when the source and destination have differences in the way they organize the data they store. Consider for example a dataset that needs to be moved from a legacy database system (source) to a modern type of storage (destination). In such scenarios, a number of challenges can arise, such as conflicting data types (e. g., float values have differing precision, character string lengths do not match), diverging structure type (e. g., relational vs. semi-structured data), or different data representation formats. To overcome these challenges, it is important to have detailed knowledge about the dataset and the source and destination systems.

Data Quality Assessment Crucial for many activities is the *quality* of a dataset. Data quality has been described as "fitness for use", i. e., "data that are fit for use by data consumers" [WS96]. To find out whether or not this requirement is met by a given dataset, the data quality needs to be assessed. This is usually done with the help of data quality metrics that quantify specific dimensions of the dataset, such as accuracy, completeness, or accessibility [PLW02]. These metrics are derived and calculated with the use of a specific methodology. Over the years, many different data quality assessment methodologies have been proposed, and a comprehensive overview of these is given by Batini et al. [BCFM09].

Data Analysis The umbrella term *data analysis* (also called *data analytics*) is used to denote a wide variety of techniques and processes for retrieving, processing, transforming, and evaluating data. As such, it is closely related to mathematics in general and statistics in particular. The purpose of data analysis is to gain insight from the analyzed data and learn something new about it. It is applied in virtually every domain where data needs to be processed, such as finance, insurance, logistics, health care, etc. Within the scope of data analysis, a number of specialized sub-disciplines have emerged. Amongst others, these are descriptive statistics [JWHT13], exploratory and confirmatory data analysis [Tho04], predictive analytics [Sie16], and text analytics [AZ12].

Data Mining Aggarwal defines data mining as "the study of collecting, cleaning, processing, analyzing, and gaining useful insights from data." [Agg15, p. 1]. As this definition is substantially close to the one of data analysis above, their relationship shall be further examined here. In 1996, Fayyad et al. wrote that "Data Mining is a step [...] consisting of applying data analysis and discovery algorithms that [...] produce a particular enumeration of patterns over the data." [FPSS96]. This suggests that the term

data analysis is used to describe the algorithms and the technical side, while data mining makes use of data analysis and applies it to a specific dataset. The same notion can be found in the work of Nisbet et al., who state that "data mining is doing data analysis (or statistics) on datasets (often large) that have been obtained from potentially many sources." [NEM09, p. xv]. Interestingly, this later definition also includes hints of the increased requirements due to Big Data (large datasets, many sources). Other authors do not make a distinction between the two terms, and prefer the usage of both terms simultaneously ("data analysis/data mining") [Mya06].

For the scope of this thesis, the precise nature of the relation between data analysis and data mining is of little interest, and thus, they shall also be treated synonymously.

Data Warehousing To support decision making within a company, decision support systems (DSS) have been developed. At the core of most DSSs is usually a *data warehouse*, which is a subject-oriented, integrated, nonvolatile, and time-variant collection of granular corporate data [Inm05, p. 31]. The data warehouse is separate from operational databases and acts as the central point from which information about past events that are important for business (e. g., sales, transactions, customer interactions, etc.) can be retrieved. The progressive form of the term, data warehous*ing*, describes all activities surrounding such a data warehouse, e. g., its construction, operation, and maintenance [CD97]. Establishing a data warehouse is essentially a data integration project that brings together relevant internal and external sources. However, to allow proper usage of a data warehouse, certain design guidelines should be considered, such as choosing a suitable schema template (*star* or *snowflake*) and carefully designing fact and dimension tables and their attributes. All in all, a thorough understanding of the input data sources is required to ensure the success of such a project.

3.3 Information Overload

In Section 3.1 it was described what happens when too little is known about a dataset and why that is a problem. This section considers the other side, i. e., what happens when *too much* information is present. This phenomenon is usually referred to as *information overload* and has been recognized very early on in the history of computer systems. For example, Hiltz and Turoff wrote in 1985 that "unless computer-mediated communication systems are structured, users will be overloaded with information" [HT85, p. 680]. It should be noted that this was at a time when the Internet was still in its infancy and the information available to a user was merely a fraction of what it is today. Since then, the issue has been studied in many ways and was given numerous names, such as *data smog* [She98], *analysis paralysis* [SC97], *information fatigue syndrome* [Opp97], *infobesity* [26], and *infoxication* [3]. All these word plays share the notion that too much information has the inadvertent effect of impeding the ability of an individual to make a good decision. Thus, the issue is a cognitive one that stems from the fact that mankind has invented machines that produce information more quickly than a human can process it [EM00]. This is a severe problem, because in order to cope with the modern world that we have created and be successful in it, new information

needs to be processed all the time [LO96]. Keim et al. give three different reasons for why information overload can occur [KAF+08]. These are:

- Information is irrelevant to the current task
- Information has been processed in an inappropriate way
- Information is presented in an inappropriate way

Systems that display data or information should be designed in a way that these problems do not occur or are at least mitigated. Depending on the context, this requirement can be arbitrarily difficult to achieve. Deciding whether a piece of information is relevant to a user is usually not possible for a computer. While there are systems that suggest certain items to a user, i. e., recommender systems [RV97], these systems cannot verify the appropriateness of their suggestions before the user reacts to them. Similarly, the appropriateness of the processing and presentation cannot be verified automatically, as it may depend on external factors such as the task or the perception of the user. For example, one user might prefer bar charts while another prefers pie charts. Displaying one or the other could be detrimental depending on the current user, while displaying both at the same time is clearly redundant and introduces even more complexity. Thus, these systems need to be flexible and dynamic and allow hiding information, inspecting the way it has been processed, and changing the way it is presented.

Information overload is a prevalent issue in data profiling. The reason for this is that a profiling tool has the capability to generate high volumes of metadata, even from small amounts of data as input. This can lead to a situation in which a user is facing more information than before, which may be detrimental to achieving data comprehension. This issue can at least be partially alleviated through the smart design of user interfaces that do not overwhelm the user with all information at once. One approach for this type of design is called *adaptive user interface*, which is based on the approach that individual differences in a user's cognitive abilities can be accommodated for with dynamic interfaces [Ben93]. Such a system infers the user characteristics from his interaction patterns and then adapts itself accordingly. Further developments in this area make use of learning algorithms to improve over time [JCK03]. Adaptive user interfaces have been successfully implemented in various domains, e. g., health care [Ram09] or education [BTN90, p. 202]. However, as of today, no major effort has been made to design an adaptive user interface for a data profiling tool.

In general, solutions to the information overload problem can be categorized into two types: system-centric and user-centric. System-centric approaches try to improve the systems that display information to avoid the issues stated above. The degree to which this is achievable depends on the data and the use case. For example, there does not exist an intuitive way to visualize four-dimensional data. Thus, they cannot be visualized effectively and remain complex in their representation.

User-centric approaches aim at educating and teaching the users to better understand what is being presented. This skill is also referred to as *information literacy*, and Edmunds et al. believe this to be the key to reduce information overload [EM00]. However, it is unreasonable to expect that a one-sided approach can solve this problem in all instances, and thus, both approaches need to be combined in an intelligent manner.

3.4 Data Comprehension

As already mentioned in Section 3.1, the term *data comprehension* is used in this thesis to denote the state at which a user knows enough about a given dataset to be able to work with it in the context of a specific task. Thus, it acts as an intermediate goal that is desirable to be achieved in situations that involve unfamiliar datasets. This section describes the key characteristics of this state and their implications.

First off, due to its definition, data comprehension is dependent on these three aspects:

1. The dataset (cf. Section 3.1)
2. The task (cf. Section 3.2)
3. The user (cf. Section 3.5)

Whenever the question occurs whether data comprehension has been achieved, these three aspects need to be considered. The task is important, because it determines the depth of knowledge that is demanded from the user. For example, if the task is to migrate a dataset from one system to another, very little information is necessary, e. g., the size and format. Further information about the data values themselves are not required, and thus, data comprehension can quickly be achieved. If however the task were different and more complex, like a data mining effort, then much more information about the dataset would be needed.

Achieving data comprehension can be difficult. Usually, it is done by gradually gathering insights about the dataset. To understand this process, it is useful to define what exactly an *insight* is. The word itself carries multiple meanings, but the most relevant here is that an insight is a piece of information. Insights are especially useful if they have not been known before, i. e., the information is new for the user, because only then can something be learned. The assumption then is that insights about datasets can be derived directly from metadata that are generated through data profiling. This can be generalized to the statement that data comprehension can be achieved through data profiling, which shall be demonstrated later on.

A core challenge that remains is to evaluate if data comprehension has been achieved, i. e., determine the point at which enough insights have been gathered. As already mentioned in Section 3.1, it is unreasonable to strive for every piece of information possible. However, it is very tricky to know when to stop due to the lack of quantifiable or objective measures. This issue is exacerbated by the many factors that influence the comprehension of the user, such as his ability to learn new facts, the complexity and requirements of the task, etc. Still, for efficiency reasons it is desirable to have a measure in place that shows whether sufficient data comprehension has been reached.

Such a measure could be implemented in the following way: A quiz-like game is set up in which the user is asked questions about the data. These questions aim at various characteristics of the data, e. g., how many rows there are, how old the data is, what it is about, etc., and get progressively harder. The user can answer through multiple choice or free-form text input. After the game, a score is calculated which directly reflects the user's comprehension of the data, i. e., the higher the better. This "data comprehension game" is especially applicable in scenarios with many users and static, non-changing data. With only few users, the effort to

set up the game would not be worthwhile, and if the data is changing frequently, too much effort would be needed to keep the questions and answers up to date.

The observed practice for evaluating data comprehension is to rely on the instinct of the user. This means that the user gathers insights about the data as long as he deems necessary and until he feels confident about his capability to commence work on the task. With this reliance on the user and his skills, this component of the system should be further investigated, which is done in the next section.

3.5 User Characterization

So far, the term *user* has been employed several times already. This section aims to give a more detailed insight into what a user is, how he can be characterized, and especially what needs to be considered in cases where multiple users interact.

In Information Systems, a user is somebody that uses any kind of system, e. g., a computer, an application, or a service. A user interacts with a system through an interface, such as a keyboard and monitor, a web browser, etc. This interaction is called *Human Computer Interaction* or *Man Machine Interaction* and due to its importance, a whole research discipline has emerged around it. A comprehensive overview can be found, e. g., by Dix et al. in [DFAB03].

However, a user is not only defined through his interaction with the system he is using. A further characteristic to consider is the goal or purpose that the user has in mind before he approaches the system. Consequently, the user needs to be aware which system is suitable to help him with the accomplishment of his goal, which in turn is dependent on his proficiency in the usage of that system. This is similar to the concept of "sociotechnical systems", which consider the trinity of human, machine and goal as the cornerstones of successful system design [LL95, p. 20]. Consider, for example, a user that wants to learn more about a dataset by using data profiling. He is confronted with a variety of multiple profiling tools that all offer a wide range of functions and features. Choosing a tool then depends on numerous factors, such as previous experiences, perceived utility, ease of use, etc.

3.5.1 Skill dimensions

In characterizing the potential users in a data-centric scenario, two dimensions shall be considered here: *domain knowledge* and *technical expertise*. These are the main dimensions that are important when assigning people to tasks in an IT context. For simplicity reasons, only two levels along these dimensions, low and high, are considered here. The first dimension, domain knowledge, describes how well the user understands the business domain to which the data belongs, e. g., finance, insurance, health care, etc. A higher level of domain knowledge can be acquired through formal education, studying the respective field, or simply acquiring experience by working with the subject matter. A user who has gained thorough domain knowledge shall be called *domain expert* (sometimes also referred to as *subject-matter expert* [13]). Technical expertise on the other hand describes a user's skill in dealing with information technology, e. g., setting up and configuring IT systems, coding, programming,

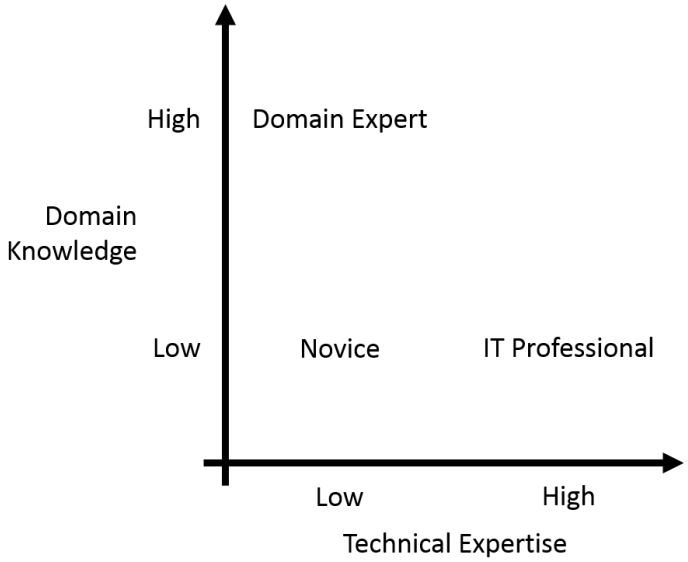

Figure 3.5: Classification of users according to skill level.

etc. Consequently, a user with a high level of technical expertise is called an *IT professional*. New users with low levels of knowledge in both dimensions are called *novices*. These different user classifications are depicted in Figure 3.5. The obvious question now is what to call users that have high levels in both dimensions. Maybe they could be called *veteran*, *master*, or *power user*, but ultimately, a concise naming is not needed, because in practice, these people are very hard to find, which is why the upper right corner is left blank in Figure 3.5. Thus, the focus in this thesis will be on the previously mentioned three types of users and their interactions and relations.

3.5.2 Multiple Users

In many scenarios, more than one user needs to be considered. For example, companies often organize their work in projects that consist of multiple project members. Within a project, tasks may be delegated to smaller teams consisting of, e. g., two to three people. When these teams jointly work on a task, they need to communicate and coordinate, and agree on the tools and systems to use.

With multiple users, a distinction can be made according to the comparability of each group member's skill set. If everybody has similar skills and knowledge, the group is homogeneous, whereas it is heterogeneous otherwise. A homogeneous group is easier to manage due to the fact that everybody can perform every task and group members are essentially interchangeable. In a heterogeneous group, some organizational challenges need to be overcome. Differences in skill levels need to be identified and tasks distributed accordingly. This can also be used as a teaching opportunity where a novice learns from an expert. However, a heterogeneous group has the advantage that it can combine its domain knowledge with its technical expertise to achieve its goal.

3.6 Common Solution Strategies

As the problem of facing an unfamiliar dataset is frequently encountered, one obvious question to ask is how people have dealt with it before. This section gives an insight into some common solution strategies that have been observed.

The most simple approach is to just start right away without any further preparation. If the dataset or the task at hand is of low complexity, this can work out successfully. However, this is rarely the case. More often, going in blind leads to backtracking when specific characteristics of the datasets are revealed during the later stages. For example, consider the task of setting up an ETL process to transfer data from a file to a database. One could just start implementation right away without looking deeper into the data and try to execute it. The lack of data inspection can lead to numerous errors, such as non-matching data types (e. g., trying to insert a string into an integer column), differing value formatting (e. g., date format not conforming), or incorrect or missing integrity constraints. Some of these errors will be caught by the DBMS right away, while others persist until they are found, if they are found at all. Correcting these errors requires going back to previous stages and revising the initial work. Thus, this approach is ill-suited for any moderately sized project.

To get at least some degree of familiarity with the dataset, the user may also a apply a technique called *data gazing* [May07, p. 154], which involves manually browsing through and looking at the data. This is a very intuitive approach that is appropriate if the size of the dataset does not justify the use of more extensive methods. However, it also suffers from poor scalability and susceptibility to errors. Also, its success is heavily dependent on the experience of the person performing the gazing.

A better idea is to include the information and metadata that may already be available. For example, the dataset may have a documentation attached that describes what it is about, how it was recorded, how it is structured, and more. Careful examination of a high quality documentation can prevent many errors and misunderstandings and is thus advisable. Still, the reality of many datasets is that no documentation is available, or that it is outdated, or of such low quality that it provides no tangible benefit. In these situations, it may be possible that there is another person with more knowledge and expertise regarding the dataset in question. Asking them about it, or maybe even including them in the project, is a worthwhile consideration.

When there is no previous source of information available, people tend to resort to ad hoc practices that involve just-in-time retrieval of needed information (e. g., about data types or formats) from the dataset using provisional methods [AGN15]. This can range from inspecting the data using spreadsheet software, over posing hand-written “quick and dirty” queries, to calculating some self-made measures on the fly. The results of these methods are mostly consumed immediately to inform a relevant decision at hand. Thus, information is usually not recorded in persistent storage, which hinders reusability and reproducibility.

The logical next step is to have standardized techniques and algorithms in place that enable the extraction and storage of necessary information about the dataset. This is precisely what data profiling tools offer. More details about data profiling tools are given in Section 4.4. The advantage of using a specialized tool over ad hoc methods is that the user can assume that the computations and calculations done by the tool are (mostly) correct and reproducible. Most

tools also offer the ability to visualize the results for easy interpretation. A core challenge that remains is that profiling tools tend to offer a vast amount of procedures that compute all kinds of information, which leads to the final approach.

4 Data Profiling — State of the Art

This chapter aims to give a comprehensive view of data profiling from an academic perspective. To this end, Section 4.1 first defines the term *data profiling*. This is followed by a typology of metadata in Section 4.2, which consists of two parts. The first part is a comprehensive overview that details which metadata types are commonly considered in data profiling scenarios, how they are defined and in which cases they are applicable. The second part of the typology presents classifications that have been proposed upon these metadata types in an effort to sort and group them together. In Section 4.3, the point of view is shifted to a higher level and data profiling is regarded from an organizational, process-oriented perspective. Finally, Section 4.4 shows how data profiling can be applied in practice by describing a selection of software tools for this purpose.

4.1 Definition

To lay the groundwork for the remainder of this thesis, this section summarizes the various points made so far and presents a definition for *data profiling*. To this end, the generic profiling model introduced in Figure 2.3 is revisited. Applying this generic model to the domain of data gives a more refined picture of the relevant tasks and how they are related. The result of this top-down approach is shown in Figure 4.1 and is subsequently described in more detail.

The primary input is the dataset, which is potentially unfamiliar to the user (cf. Section 3.1). The secondary input is external metadata, which comprises every piece of information that is not part of the dataset itself, such as documentation, provenance information, previously discovered metadata, etc. This type of information, its possible sources and means for extraction are explored later in Chapter 6.

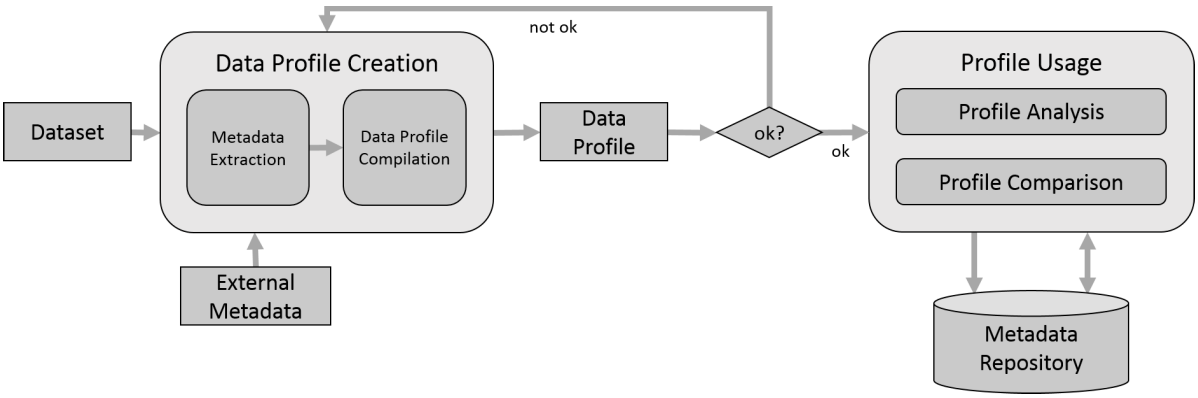

Figure 4.1: Profiling activities model applied to data profiling.

After the collection of the inputs, the creation of a data profile commences, which consists of metadata extraction and data profile compilation. The extraction of metadata is a core activity that has been well researched in the past [AGN15]. It involves first a declaration of which type of metadata is wanted. Broadly speaking, metadata is "data about data", i. e., data that is used to describe other data and its characteristics. More details about metadata and their typology will be given in Section 4.2.1. Second, an algorithm for its calculation must be selected. Some of these algorithms have such a high computational complexity (e. g., functional dependency detection is exponential in the number of attributes [Nau13]) that the use of a heuristic makes more sense, which sacrifices accuracy for a reduced runtime complexity. For many types of metadata, multiple algorithms and heuristics exist, which vary in their efficiency and applicability.

Once all required metadata have been extracted, they can be compiled into a data profile. Common techniques for this purpose range from simple tables that show numerical results to sophisticated visualizations with various types of charts. This compilation procedure is not standardized, which can lead to varying forms of representations across different tools. This issue is linked to information overload as presented in Section 3.3. Once compiled, the data profile is inspected manually by the user to check whether it needs to be refined. If this is not the case, the usage phase commences.

Profile usage in data profiling most often entails a manual analysis of said profile immediately after its creation. This means that the user carefully examines the results of the previous phase and looks for insights that help him achieve data comprehension (cf. Section 3.4). This procedure is based on the assumption that insights about a dataset can be learned from its metadata, which are usually smaller in volume and thus, easier to digest and interpret. In many cases, the data profile is discarded after it has been analyzed. However, it is usually a better idea to store it in a specialized metadata repository. This obviates the need to redo the whole process if the same data profile is required in a future project by allowing re-usability of results.

Storing data profiles also enables the second activity for data profile usage, which is comparison. This activity is less frequently performed in data profiling, but has two major use cases in which it can play a crucial role. The first is the assessment of compliance with standards, mostly used in the context of data quality [PLW02]. Here, the profile in question is compared with established or expected norms about data quality metrics. For example, if a requirement is imposed that a dataset may have at most 10% null values, it can quickly be checked through comparison whether this requirement is met or not. This procedure can also include more complex rules and automated comparison and notification. The second use case for profile comparison is tracking of change over time, also called *database evolution* [TS92]. Here, the profile is compared with older profiles of the *same* dataset. This is useful when assessing whether specific changes to, e. g., data management policies or procedures have had the desired effect. Essentially, this is a form of data monitoring on a higher, aggregated level, i. e., on the metadata level.

All these considerations can be summed up as follows:

> "A data profile is a collection of metadata that represents a dataset in a concise way. Data profiling is an activity conducted by a user on a dataset and its external metadata to create a data profile that can be used for analysis and comparison purposes."

The second half of this section provides definitions for data profiling as proposed by other authors. These definitions reflect how these authors emphasize different aspects of data profiling and what it entails. A discussion of these individual differences helps to round out the overall concept.

Naumann: Data Profiling Revisited, 2013 In 2013, Felix Naumann wrote a survey about data profiling and the current state of research and literature about it. As the introductory sentence, he cites Wikipedia which provides the following definition:

> "Data profiling is the process of examining data available from an existing information source (e. g., a database or a file) and collecting statistics or small but informative summaries about that data." [Nau13, p. 40][42]

There are several overlaps with the previously developed definition: First, data profiling is described as a process which is similar to an activity as they both describe actions that are being performed. The person doing the profiling is, however, not further characterized in this definition. The next similarity is the input data, which is described as "data available from an existing information source" and thus, largely corresponds to the dataset as introduced in Section 3.1. Note that so far the existence of available data as input is a prerequisite to the profiling process. Later on, it will be discussed what can be done if this condition is not met.

The Wikipedia definition further specifies that the results of data profiling are "statistics" and "summaries", which are a subset of metadata as mentioned before. This particular choice of words gives a more tangible image of what exactly is being produced during profiling, and also hints at why it is done, i. e., provide a short abstract of the data.

Abedjan et al.: Profiling Relational Data: A Survey, 2015 Abedjan et al. published a paper in 2015 titled "Profiling relational data: a survey" [AGN15] with the goal to classify data profiling tasks and provide a mostly algorithmic overview over the state of the art for each task. It should be noted that Felix Naumann has co-authored this paper. In their work, the authors propose the following definition of data profiling:

> "Data profiling is the set of activities and processes to determine the metadata about a given dataset" [AGN15, p. 557]

Whereas previously only a single activity has been considered, this definition increases the multiplicity to a "set of activities". However, there is virtually no difference in referencing each individual step as data profiling versus collectively calling all efforts by that name. Thus, the implications of this distinction are largely non-existent. Lastly, the input and output are described as "a given dataset" and "metadata" respectively, which is identical to the definition from the previous section.

Olson: Data Quality: The Accuracy Dimension, 2003 A book called "Data Quality: The Accuracy Dimension" has been published by Jack E. Olson in 2003 [Ols03]. It consists of three parts: (I) Understanding Data Accuracy, (II) Implementing a Data Quality Assurance Program, and (III) Data Profiling Technology. It is the third part that has a high relevance to the contents of this thesis. Furthermore, the author is called "one of the original developers of data profiling technology" on the backside of the book. This makes it a useful, albeit slightly dated, resource on the matter. In the book, Olson uses the following definition:

> "Data profiling is defined as the application of data analysis techniques to existing data stores for the purpose of determining the actual content, structure, and quality of the data." [Ols03, p. 122]

Here, data profiling is described as an "application of [...] techniques" instead of an activity or a process. This embodies the perception that data profiling does not have any techniques on its own, but rather it uses techniques from another domain (here: data analysis) and applies them in a different context. This perception can be described as outdated as by now, specialized data profiling algorithms and methods have been developed (cf. Chapter 5 and [AGN15]).

The next part in Olson's definition is "existing data stores". This is a direct correspondence to the previously described "dataset" and shows once more that the existence of data as an input to the profiling process is essential. In the last part, the purpose of data profiling is characterized as "determining the actual content, structure, and quality of the data". In a sense, this goes deeper than just the discovery of metadata by emphasizing to which end the profiling results are used. Still, there is an implied similarity to the analysis purpose of the working definition, because structure and quality are part of the analysis output.

It is interesting to note that Olson deliberately uses the term "actual content", because it implies the existence of *non-actual* content, i. e., supposed or perceived content. This underlines the fact that Olson recommends to start with the assumption that everything that is assumed to be known about a given dataset is wrong, and only thorough analysis with data profiling technology can reveal the *actual* content. This assumption calls for a complete run through the entire process, and is thus, very expensive and pessimistic. It can be argued that there are cases in which preexisting knowledge about data (e. g., in the form of metadata or documentation) is trustworthy and thus, should be used without costly reevaluation.

4.2 Metadata Typology

Having established that one of the purposes of data profiling is the extraction of metadata, it shall be explained in more detail what exactly metadata is in this context and what it is comprised of. To this end, Section 4.2.1 first provides a broad overview of commonly encountered types of metadata, how they are defined and calculated, as well as how they can be useful in a data profiling scenario. After that, Section 4.2.2 describes various classification schemes for metadata that have been proposed in the past. Lastly, a process to operationalize the extraction of metadata is proposed in Section 4.2.3.

4.2.1 Common Types of Metadata

The term *metadata* is composed of the two terms *meta* and *data*. *Meta* is a Greek word that can roughly be translated to *after* or *beyond*. Modern usage of the term as a prefix to another word has shifted more towards *about* in order to indicate an abstraction from the word onto a higher level. Thus, metadata is essentially data about data [HT03, p. 9], or as the Merriam-Webster dictionary puts it: "data that provides information about other data" [18]. This also underlines the intuition that looking at metadata is useful for understanding a given dataset due to the descriptive nature of the former. The same understanding is held by the International Organization for Standardization, which declares that "metadata is defined to be data that defines and describes other data" [ISO04, p. 10]. However, in a later draft of the same standard, the authors also concede that this "data about data" notion is incomplete and potentially misleading. Their argument is that the term *metadata* is also frequently used to describe other things that are not data, i. e., real-world objects such as books or movies. For example, the metadata of a movie could include its release date or its director. This kind of data is data about the movie, and not about some other data. Thus, the term *metadata* is ambiguous and semantically overloaded. For the scope of this thesis however, whenever metadata is discussed, the intended meaning is that of "data about data".

The rest of this section shows what types of metadata are frequently encountered in data profiling scenarios. There is an obvious bias towards metadata that is relevant in relational databases. The reason for this bias is that this is the data model that is used by most of the literary work that deals with data profiling. Ideas on how non-relational data can be profiled are given in Section 8.2.

The following set of metadata types has been derived from two main approaches: First, various data profiling tools have been examined to screen the features they offer. In particular, the tool *Talend Open Studio for Data Quality* has been a notable source due to ease of access and its wealth of functionality. Second, the body of knowledge about data profiling was reviewed to find out which metadata types receive the most attention. Here, one notable source is the work by Abedjan et al. in [AGN15], which provides a good overview on data profiling in general and metadata types in particular. The results from both approaches have been merged and unified to present one comprehensive set. However, note that the set of metadata types described here cannot and does not claim to be complete, because there is an unlimited number of metadata types conceivable. Technically, anybody can invent arbitrary rules to define any types of metadata that are relevant for only specific use cases, which raises the number of metadata types to infinity. Instead, the goal is to provide an overview of those types of metadata that are pertinent to a wide range of use cases and scenarios.

For reasons of clarity, the following list is split into two parts. The first part deals with metadata types that are generally applicable to a single field or column. In the second part, more complex types of metadata that involve multiple fields or columns are described. At the end of each part, a tabular overview recapitulates the metadata types for easier reference. Finally, the case of metadata that spans more than a single dataset is addressed in a separate subsection.

Table 4.1: Sample CloudHost employee dataset: "emp".

id	name	age	department	hours
1	John Smith	38	HR	39.75
2	Sun Li	42	HR	39.75
3	Jane Miller	⊥	HR	19.875
4		45	Sales	40

Single Field

To illustrate the descriptions in this section, all metadata types will be evaluated for the CloudHost employee dataset shown in Table 4.1, which was already used back in Section 3.1. This dataset is meant to represent an excerpt from CloudHost's employee database and consists of the five columns `id`, `name`, `age`, `department` and `hours`. When applicable, generic SQL code samples are given to show how the respective metadata can be calculated. These code samples will use the placeholder names `tbl` for a table and `col` for a column, and have been tested with MySQL server version 5.6.21.

Name. A name is any label that allows identification of and reference to a specific object or entity. Names are usually represented as alphanumeric strings and are an integral part for the structure of any dataset. They are used to distinguish its different parts and components. Most prominent is the name of a field or column, because it is the primary way to uniquely specify the part of the data that is relevant for a certain operation. In the example case, the five columns have their respective names explicitly spelled out in the first row. This means that the first column is named *id*, the second column *name*, and so on.

Names are also used to identify other parts of the data, like tables, databases, graphs, or similar concepts depending on the data model. In most relational DBMS, the name of a column must be unique within one table. Otherwise, it would not be possible to distinguish two same-named columns. To differentiate between two same-named columns in different tables, the table name and a separating dot are commonly used as a prefix. Thus, `id` could also be referred to as `emp.id` in order to differentiate it from another column `id` that might exist in a different table.

It can be challenging to handle cases in which the names of the columns are not explicitly provided, e. g., when data is loaded from an undocumented CSV file. Most tools handle such cases by using placeholder names with ascending suffixes, such as `COL_01`, `COL_02`, etc. These should be replaced by proper names as soon as possible, because having descriptive names in place is a very important part in making data understandable. Thus, when naming columns (or tables or graphs...) one should always try to describe the content in a concise manner that allows others to easily understand what is meant. If applicable, naming conventions should be followed. For example, Oracle proposes a set of naming standards for all objects within a database [23].

At CloudHost, most datasets are stored in relational databases, where names of tables and columns are enforced to be uniquely identifiable. Combined with a well drafted policy of how

these names should be designed, it is usually very easy to understand a given CloudHost dataset. However, when exporting datasets out of the database, it is very common that tables and columns are renamed or merged in non-unique fashion.

In one particular case of the original CloudHost project, a dataset needed to be processed, which was the result of a larger join query that combined the data from many different tables. That dataset was provided in the format of a spreadsheet file, without any constraints regarding the existence or duplication of any names. As such, that file contained four distinct columns with the exact same name `ID`, with no further information as to what data each of these columns contain or how they should be interpreted. As it turned out, each of these columns referred to a different source table, and it required a lot of manual back and forth between different involved stakeholders to resolve this ambiguity and establish a mapping. If these `ID` columns had been renamed uniquely to include a reference to their respective source table, this problem would not have existed.

In another case, a CloudHost analyst from Denmark created a report in which he translated every column name to Danish. Sharing that report with his other Danish colleagues did not cause any issues, but when the report was sent to the German headquarters for verification, nobody could understand what any of the columns meant. Thus, this case also required unnecessary additional work to fix a problem that was caused by naming issues.

The lesson here is that names are an important part of any dataset and should be treated carefully. Renaming a column can sometimes save time as in the first case, or cause additional problems as in the second case.

Data Type. The data type of a column specifies the type of values that are allowed in it. The most general data type is *character string*, which allows a sequence of characters from a fixed alphabet (e. g., the American Standard Code for Information Interchange, ASCII) to be stored. In the example table, `John Smith` is a character string. By putting further restrictions on the value space, other data types can be derived. For example, if only numeric digits are allowed and no letters or other characters, *integers* can be represented, like the age of John Smith "38". With the addition of a decimal separator, the data type *decimals* is expressible, e. g., the hours of John Smith "39.75", which uses a dot as separator.

Note that these considerations are based on a purely representational view, that is, what the data looks like to a user. The internal representation of the different data types on an actual (database management) system is much more complicated and involves technical details that are omitted here as they are not relevant for profiling purposes.

Further data types, like *time* or *date*, are constructed by enforcing a specific pattern and/or value ranges that correspond with the desired output. For example, the pattern `YYYY-MM-DD` specifies that a date should be stored by using the first four digits to represent the year, then a dash, then two digits for the month, another dash, and lastly two digits for the day of the month. The underlying system could further require that the value of MM is between 01 and 12, or DD is between 01 and 31. Lastly, for data that has no useful representation in the form of characters and numbers, a *binary* data type is usually employed.

When working with a concrete DBMS, it is customary to refer to the data types not by their generic names, but rather through DBMS-specific identifiers. In most SQL-based

implementations, these are `CHAR` and `VARCHAR` for character string, `INT` for integers, `FLOAT` or `DOUBLE` for decimals, `TIMESTAMP` and `DATE` for time and date, and `BLOB` (short for *B*inary *L*arge *OB*ject) for binary data. Most of these data types also have variations for differing sizes, like `TINYINT` or `MEDIUMBLOB`.

The data type of a column is a crucial information that is useful in many different ways. For example, if it is known that two columns have incompatible data types (say, *string* and *number*), then it makes probably no sense to attempt to perform a join operation along those columns. Similarly, instances where a column should be of one data type, but includes values of another type (e. g., a *string* in a *number* column), can be flagged as violations that should be looked into.

Ideally, the data type of every column is given as part of the schema or documentation and does not need to be inferred or determined manually. However, this is not always the case, e. g., when data is dumped into a plain text format like CSV. Then, it is usually not possible to unambiguously find the data type of every column. For example, the column `id` in the sample dataset consists of only integers, so it would be natural to assume integer as the data type. However, this is not the only possibility, because this column could also be declared to be of type *string* without causing any violations. Thus, domain knowledge might be required, like the information that the `id` column is used as the primary key which performs faster on integer types. In conclusion, an automated data type determination should always be followed up by a manual inspection by a domain expert.

Row Count. The number of rows is a very basic type of metadata for a dataset. It is usually given on a per-table basis, i. e., every table has its own distinct count. The row count is a very useful number to have when assessing the size of a dataset. Due to its simplicity, almost every data-centric tool automatically calculates and displays this number. In cases where the term *row* does not fit with the structure of the data, a synonym is used, e. g., entity count, node count, or relation cardinality. Sometimes, it is also referred to simply as the size of the dataset, although this can be misleading due to similar names of unrelated concepts, such as the space in bytes allocated to the dataset.

The row count for a table `tbl` can be calculated in SQL using this query:

```
SELECT count(*) FROM tbl
```

This query returns 4 for the CloudHost employee dataset. Note that the header row is considered part of the schema and is thus not interpreted as a data row. This distinction is important in cases where it is not clear whether the first row should be interpreted as the header row, which happens often in plain text formats, e. g., CSV files.

One major use case in which the row count plays a crucial role is query optimization. A query can be optimized, i. e., its execution sped up, by transforming it into an equivalent query with less costs [Cha98]. This is achieved through re-arrangement of processing steps in such a way that intermediate results are minimized. Knowing the row count of every participating table of a query is a prerequisite for this procedure.

Another use case for the row count is the first step of validation after data migration. Recall that data migration is act of moving data from one location to another Section 3.2. There

are numerous reasons why such a migration might be incomplete: data being unreadable due to corruption, data getting lost during transmission (e. g., due to packet loss or other network issues), or data being unwritable at the destination due to integrity constraints or other restrictions. How these errors are identified and treated depends on the implementation of the system performing the migration. Some systems may give a detailed report of any encountered error, while others may silently ignore them, resulting in data loss. In any case, it is reasonable to at least compute and compare the row count at every step along the way, to make sure no data is inadvertently lost.

In the CloudHost scenario, one step of data processing consisted of the data being pulled from a source system, stored as a spreadsheet file, compressed into a ZIP archive, sent via mail, uncompressed, and loaded into a target database. It is very inexpensive to compute the row count of the dataset at three points during this procedure: in the source system, the spreadsheet file, and the target system. Comparing the numbers revealed that, indeed, a sizable number of rows got lost between the file and the target system, which was likely caused by an error in the compressing and uncompressing steps. Redoing these steps fixed the issue and corrected the row count to be the same in the source and target system.

Null Count. Most (relational) databases support a special value called *null* to mark cells whose true value is not applicable or not known. The *null count* can be computed for a column by counting all these occurrences. A simple SQL query to do so is for example the following:

```
SELECT count(*) FROM tbl
WHERE col IS NULL
```

Note that the `WHERE` condition uses the special operator `IS NULL` instead of a direct comparison, i. e., `col = NULL`. This is because null is not a value, but rather the absence of one, and thus cannot be used in this way for a comparison.

In the employee example dataset, a null value is indicated with the special symbol $\perp$. The `age` column has one such value, so the null count for this column is 1. Put into context, this means that the age of Jane Miller is unknown. This observation can be used in different ways. For example, the completeness of a column can be defined as the ratio of non-null to total values. This is already a useful data quality metric, albeit a very simple one. Another example is that columns with a null count greater than one can be filtered out when looking for key candidates, because keys usually do not contain nulls.

During the CloudHost scenario, an analysis was conducted that focused on one specific column, which contained descriptions of the product portfolio. The results of this analysis were met with skepticism from the domain experts ("this can't be right"), and consequently, much time was spent on going through the individual steps of the analysis. In the end, it was found that the root cause for the wrong results was not in the analysis, but in the data itself: the descriptions column had a completeness of less than 10%, i. e., more than 90% of the values were `NULL`. This made the analysis mostly worthless. In this case, a quick look at the null count could have saved time and effort.

The fact that null values only exist in systems that support them (i. e., database management systems) has a number of ramifications. For example, it is common practice to transfer data

by exporting into a flat-file format and re-importing into the target system. However, as the flat-file format does not support null values explicitly, they are usually converted into empty values. This makes them indistinguishable from actual empty values. During the re-import, a decision needs to be made whether empty values should be converted back into null values or kept as empty values. No matter how that decision is made, information will be lost.

Blank Count. A blank value is a value that is empty, i. e., it contains an empty string with no characters and length zero. This is most commonly expressed by writing two quotes with nothing in between (‘’). Blank values are different from null values in systems that support them, because they are treated as actual values which allows the usage of standard comparison operators. Furthermore, this implies that a column with a `NOT NULL` constraint may very well contain blank values.

Counting the number of blank values in a given set of values yields the blank count. In an SQL query, the blank count of a column can be computed as follows:

```
SELECT count(*) FROM tbl
WHERE col = ''
```

In the sample employee dataset, this query returns 1 for the column `name` and 0 for all other columns.

It should be noted that sometimes the term *blank space* is used to denote a whitespace character (‘␣’), which is produced by hitting the space bar on a standard keyboard. This should not be confused with the usage here, where *blank* is a synonym for *empty string*. This also implies that the blank count can only be meaningfully computed for columns with a string data type. For all other data types, the blank count will always be zero.

There is another peculiarity regarding blank values that users should be aware of. Depending on the implementation and configuration of the tool and database used, trimming functions may be applied that automatically remove leading or trailing whitespaces. This may lead to unexpected situations in which a comparison like ‘’ == ‘␣’ may evaluate to true.

Default Value Count. Many systems allow the definition of a default value for a field or column. This default value is used whenever a row is inserted without an explicit value for that column. The default value count can then be computed by counting the number of values that are equal to that default value, which translates into the following SQL query:

```
SELECT count(*) FROM tbl
WHERE col = '<default>'
```

The placeholder `<default>` needs to be replaced by the determined or assumed default value. This implies that without any pre-existing knowledge about the default value, this count cannot be computed.

In the example case, it shall be assumed that the default value for `age` is null. Thus, the query above returns 1 for that column.

Distinct Count. Two values are considered distinct if they are not equal to each other. The distinct count is the number of values in a given collection that are distinct from each other. The naming is derived from the SQL operator `DISTINCT`, where it can be used as follows:

```
SELECT count(distinct col) FROM tbl
```

In the CloudHost employee dataset, this query returns 2 for the column `department`, because there are two distinct values in it (HR and Sales).

The computation of the exact distinct count for a given dataset is straight-forward, but uses an amount of memory that grows linearly with the size of the dataset. This may be prohibitive in scenarios with large datasets or small amounts of available memory. Thus, a number of approximation algorithms have been proposed that aim to reduce the memory usage and to a lesser extent, the total runtime. As the number of distinct values in a dataset is also called its cardinality, these algorithms are commonly referred to as cardinality estimation algorithms.

A recent and comprehensive evaluation of twelve cardinality estimation algorithms is given by Harmouch and Naumann in [HN17].

Unique Count. A value in a collection is said to be unique if no other value in that collection is equal to it. In other words, a unique value appears exactly once. The unique count is consequently the number of values in a collection that are unique. Computing this number is not directly supported by a standard SQL operator. However, a sub query can be used to compute the unique count in the following way:

```
SELECT count(*) FROM (
  SELECT * FROM tbl
  GROUP BY col
  HAVING count(col) = 1)
AS unique_count
```

The inner clause selects all values that appear exactly once, and the outer clause counts the number of values returned by the inner clause. Together, this accomplishes the task of computing the unique count of a given column. In the sample employee dataset, this query returns 1 for the column `department`, because there is only one value that appears exactly once (HR).

Dividing the unique count by the row count results in the unique ratio, i. e., the percentage of values that are unique. If the unique ratio has a value of 1, then every value is unique, and the respective column is a potential primary key candidate.

During the CloudHost scenario, a situation emerged where the primary key of a dataset in a spreadsheet file needed to be determined. Spreadsheets usually do not contain information about primary keys in a dataset, so a manual search was conducted. By computing the unique ratio for each column, and then discarding all columns with a unique ratio of less than 1, the search field could be narrowed down considerably. The remaining columns were considered primary key candidates from a technical perspective, and together with the domain experts that know about the business meaning of each column, it was very easy to find the one column that should be declared as primary key.

Duplicate Count. Two values are considered duplicates of each other if they are equal. One might be inclined to think that the duplicate count then is the number of duplicates in a collection. However, this declaration is too vague as it is not clear whether *both* duplicate values should be counted or just one of them. The problem gets more complicated if more than two duplicates exist for one value. An examination of different data profiling tools reveals that different data profiling tools implement the duplicate count in different ways. The following three definitions can be distinguished:

1. The number of distinct values that appear more than once, i.e., the number of values that have at least one duplicate. The number of duplications for one specific value does not matter. This is the interpretation of Talend.

```
SELECT count(*) FROM (
  SELECT * FROM tbl
  GROUP BY col
  HAVING count(col) > 1)
AS dup_count
```

2. The number of values that are duplicates of already seen values. This is the number of values that can be considered redundant and, e. g., flagged for removal during duplicate detection and is reported, e. g., by the ataccama Data Quality Analyzer.

```
SELECT sum(c) FROM (
  SELECT count(col)-1 as c
  FROM tbl
  GROUP BY col
  HAVING count(col) > 1)
AS dup_count
```

3. The number of total values that appear more than once. While this interpretation has not been observed in any tool, it is easy to make a reasonable argument for it.

```
SELECT sum(c) FROM (
  SELECT count(col) as c
  FROM tbl
  GROUP BY col
  HAVING count(col) > 1)
AS dup_count
```

To illustrate this with an example, consider the `department` column in the CloudHost employee example, which consists of four values: $HR, HR, HR, Sales$. Counting the number of duplicates in this column can be done in three different ways. According to (1), the result is 1, because only HR appears more than once. However, it could also be argued according to (2) that the result is 2, because two instances of HR are duplicates of one original HR. Finally, the answer to the duplicate count could also be 3, because three instances of HR can be observed, and they are all duplicates because they are not unique (3). In this line of thought, the original HR can not be distinguished from its duplicates. It should be noted that

despite these different methods of counting, all three definitions agree on the result zero if no duplicates are present. Thus, there is no ambiguity to the statement "there are no duplicates in this set", but it is unclear what is meant when x duplicates are asserted if the method of counting is not specified. This is especially relevant in scenarios where data quality metrics or scores are constructed based on the count of duplicates.

The duplicate count as discussed here is only defined for exact matches of values. Depending on the use case, it can also be worthwhile to extend that notion and look for close matches, which is known as *fuzzy matching*. Independent of how the duplicate count is defined, it is usually the goal to minimize it, because a duplicate is usually an indicator for a data quality issue. The process of finding and eliminating duplicates is called *deduplication*.

Value Length. The length of a value is defined as the number of characters it contains. For example, the length of the string `John` is four, because it has four individual characters. As this concept of length applies only to a single value, it is not really useful as metadata by itself. Usually, an aggregation function is applied to derive more interesting information about a set of values (e. g., a column). Examples for these aggregations include minimum, maximum and average.

To compute the minimal value length of a column using SQL, the following query can be used:

```
SELECT min(length(col)) FROM tbl
```

In the CloudHost employee dataset, this query returns 0 for the `name` column, because the blank value in row 4 is counted as zero length. Similarly, the maximal or average value length can be computed with the built-in SQL functions `max` and `avg`. Applied to the employee dataset, the maximal value length of the `name` column yields 11. Note that the longest value is `Jane Miller` in row 3, and that counting characters also considers white spaces.

Using the average function on the `name` column results in 6.75. This can easily be verified: `John Smith` has length 10, `Sun Li` has length 6, `Jane Miller` has length 11, and the empty value in row 4 has length 0. Summing these numbers up and dividing them by their count equals 6.75, i. e., $(10 + 6 + 11 + 0)/4 = 6.75$.

These various aggregations of value lengths have a number of uses, which are typically dependent on additional domain knowledge about the types of values that are stored in a field or column. For example, if the minimal value length equals zero, there is at least one value that is empty. This hints at data quality issues when empty values are not expected, e. g., in a name column. Similarly, a minimal value length of one or two in a name column is suspicious and warrants further investigation. The maximal value length on the other hand can hint at abnormal values and outliers on the other end of the spectrum. If for example the maximal value length in a column `city` is 50 or longer, it is possible it has been misused to store additional information, like country or continent. The maximal value length also has an important technical use case. Most relational DBMSs handle string columns with an explicit maximum number of allowed characters, e. g., `VARCHAR(255)` indicates that no string longer than 255 characters may be put into that column. Knowledge of the maximal value length of a given column can thus be crucial when designing a database table to hold the data, e. g., in a data migration scenario.

Number of Decimals. In numeric data types that represent non-integer numbers, such as *float* or *double*, the decimal part is the fraction that is not integer. In colloquial terms, this can be explained as "everything after the dot". For example, the decimal part of '1.23' would be '23'. The number of decimals then is defined as the length of the decimal part, i. e., 2 in the aforementioned example.

Calculating the number of decimals for a given number can be done by stripping away the integral part ("everything before the dot") and then counting the number of digits that remain. In SQL, the following query calculates the number of decimals for every value in a given column:

```
SELECT
  col,
  GREATEST( LENGTH(CAST(col - FLOOR(col) as CHAR(50)))-2,0)
    as decimals
FROM tbl
```

The subtraction of two is done to account for the leading zero and the decimal dot, which is a MySQL-specific correction that may need to be adapted for other DBMSs. This does however lead to negative results when numbers without a decimal part are put in, which is unwanted. Thus, the `GREATEST` function catches these cases and sets the result to zero. This query can be applied to the *hours* column of the employee dataset. The results are 2 for '39.75', 3 for '19.875', and 0 for '40'.

Similarly to the value length before, it is usually not desirable to look at this result for each individual value. Instead, it should be aggregated in a way that makes semantic sense for the data, e. g., using `min`, `max` or `avg`. For example, in a scientific experiment where data is generated through physical measurements, the number of decimals is an indicator for the precision of the sensors. Looking at the minimal and maximal number of decimals provides insight into the precision range of measurements and allows its validation. Note that in this example, it might also be advisable to remove trailing zeros before counting decimals to assess the true informative content, and not be misled by padded numbers.

Value Statistics. These types of metadata are computed using simple statistical functions applied to the values of an input dataset. Starting with the lowest complexity, the *minimal* and *maximal* values of a dataset are the lowest and highest values, respectively. SQL has built-in functions that allow a direct computation of a column's minimal or maximal value. The following query returns the minimal value:

```
SELECT min(col) FROM tbl
```

Applied to the `age` column of the CloudHost employee dataset, this query returns 38. Replacing `min` with `max` returns 45 as the maximal value. Note that these functions are defined on the basis of an underlying order of the data type in question, which is also used when sorting the data. For numerical data, this order is trivial. For date data types, like `datetime` or `timestamp`, the convention is that older dates have a lower value and newer dates a

higher value. When the data in question contains string data, it depends on the system implementation which order is chosen. Most implementations apply an alphabetical order, such that, e. g., 'A' is treated as a lower value than 'B'. However, other implementations could also use a different order, or even disable these functions for string data types entirely.

The next concept is the *average*. Mathematically speaking, there exist numerous differing definitions of what exactly the average of a collection of values is. In most cases, the term refers to the arithmetic mean, i. e., the sum of the numbers divided by the count of numbers. Whenever the term *average* is used in this thesis without further specification, the arithmetic mean is meant. The definition of the average already reveals the constraint that it is usually only defined for numerical data types. Most SQL-based systems implement an `avg` function to quickly calculate the average of a column:

```
SELECT avg(col) FROM tbl
```

For the `age` column in the CloudHost employee dataset, this query returns 41.66 for the `age` column. The calculation to verify this result is: $(38 + 42 + 45)/3 = 41.66$. Note that the null value in row 3 does not factor into this calculation at all, because it is not treated as a value.

The minimal, maximal and average value, provide a first intuition about the distribution of the data. Some tools also offer the calculation of the *range*, which is defined as the max value minus the min value. In some scenarios, a user may have a premonition or rough expectation about the values of these metadata types. If that is the case, these premonitions should be compared to their actual values, because any discrepancies here constitute valuable insights.

As said before, *average* has multiple definitions. For the purpose of data profiling, a second definition can be of importance, namely the *median*. The median of a set of numbers is the value that separates the lower half from the upper half. Note that 'lower' and 'upper' here imply that the set is sorted. Use cases of the median are very similar to the average (i. e., the arithmetic mean). Both are useful to derive the central tendency of a set of numbers and get a first insight into its distribution. The median is more robust with regard to extreme values and outliers, whereas the average is more intuitive and easier to interpret.

x-th Percentile. Another way to quickly describe the distribution of a set of numbers is the x-th percentile. The x-th percentile states the threshold value below which x% of the numbers are found. For example, the employee dataset has three numbers in the age column: 38, 42 and 45. The 33rd percentile is the cut-off age below which 33% of the employees are. The value of the 33rd percentile in this example is 38. Consecutively, it can be derived that the remaining 67% of employees are above that age. Note that following this definition, the median can also be interpreted as the 50th percentile. Further percentiles with special names are the lower quartile and the upper quartile, which correspond to the 25th and 75th percentile, respectively. The *inter quartile range*, defined as the difference between the upper and lower quartile, is a popular measure for noisy data, because it is more robust to outliers than, e. g., the range.

Mode. The next metadata type is the mode (or modus), which is defined as the most frequently occurring value in a set of values. The mode can be derived using the following SQL expression:

```
SELECT col FROM tbl
GROUP BY col
HAVING count(*) = (
  SELECT count(*) FROM tbl
  GROUP BY col
  ORDER BY count(*) DESC
  LIMIT 1)
```

In the CloudHost employee dataset, this query returns 'HR' for the `department` column, because it occurs three times, which is more than any other value. Note that the mode is not necessarily unique, i. e., there can be multiple values that occur the most.

In statistics, the mode is often compared to the average and the median, because they fulfill the same purpose of describing the central tendency of a distribution. While the detailed intricacies of the differences of these three measures are omitted here, one key purpose of the mode shall be highlighted. The mode is defined for arbitrary data types, i. e., numeric or non-numeric. In particular, it does not require any order of its input values, unlike the average and the median. Thus, in cases where there is no order of the data values, or where the assumed order (e. g., alphabetical) is not meaningful, the mode should be favored. For example, in a column that contains city names it makes little sense to ask for the average city. However, the most frequently occurring city, i. e., the mode, can provide valuable insight.

Value Frequency Table. In statistics, the frequency is the number of times a thing of interest has been observed or occurred. The value frequency is thus the frequency of a data value within a given dataset. For each value, the frequency can be computed in absolute terms, or relative to the number of total values in the set. Usually, the value frequency for multiple values (or all of them) is displayed in the form of a table, hence the name *value frequency table*. Such a table has a column for the value and additional columns for the absolute and/or relative frequency, depending on the use case. If the number of distinct values is too large, the number of rows in the table can be limited to a fixed number.

In SQL, a value frequency table with absolute frequencies can be computed using the `GROUP BY` operator. Combined with `ORDER BY` and `LIMIT`, the `x` most frequent values and their absolute frequency are retrieved with the following query:

```
SELECT col, count(*) FROM tbl
GROUP BY col
ORDER BY count(*) DESC
LIMIT x
```

Executing this query on the `department` column of the CloudHost employee dataset gives the result shown in Table 4.2.

Note that if the *least* frequent values are sought after, e. g., to look for outliers, then the order can be reverted by changing `DESC` to `ASC`. Depending on the tool used, the most or least frequent values are also referred to as *top x* or *bottom x*, respectively.

Due to their definition, value frequency tables tend to be clear and compact in their representation, which enables their quick assessment by a user. They are especially well

Table 4.2: Value Frequency Table of the `department` Column.

department	count(*)
HR	3
Sales	1

suited for data that is categorical, like the `department` column in the example case. However, value frequency tables are less suitable for fields or columns that contain many distinct or unique values, or continuous numerical variables. For example, in a primary key column where every value is unique, the frequency for all values is equal to one. In such a scenario, no useful insight can be gained from a value frequency table.

Pattern Frequency Table. Patterns are a way to abstract a concrete value into a more generalist form, which allows easy identification of prevalent structures and commonalities in, e. g., a collection of strings. There are various ways and languages to define a pattern. Many profiling tools offer a simple approach, in which the pattern of a data value is derived by going through it character-wise and replacing each character with a placeholder for that character. For example, upper-case characters are replaced with 'A', lower-case characters with 'a' and numbers with '9', while special characters are left as they are. With this, the strings `Tom` and `Jim` are both turned into `Aaa`, i. e., they adhere to the same pattern.

The values in any given dataset may have a number of different patterns. Aggregating these patterns and counting their frequency results in a *pattern frequency table*, which is similar in structure to a value frequency table, but lists patterns instead of actual values.

Pattern frequency tables are very powerful when assessing string data where a correct pattern of fixed length can be explicitly stated. For example, consider a scenario where zip codes are stored in a column. In Germany, all zip codes need to consist of exactly five numbers. Consequently, every valid zip code needs to conform to the pattern '99999' and every other observed pattern is likely to stem from some data quality issue.

Pattern Matching Count. There are many more patterns that can be observed in a dataset but require techniques more sophisticated than the simple replacement steps described above. For example, a valid e-mail address consists of letters, and numbers, followed by the @ symbol, followed by some more letters or numbers, then a dot, and finally a top-level domain indicator. This is a simplified pattern for e-mail addresses and the full specification can be found in [32]. Due to the varying length of the individual parts, a pattern frequency table is insufficient to accurately distinguish valid and invalid e-mail addresses.

This task can adequately be handled with the use of *regular expressions*, which define a precise syntax to represent patterns. They go back to the work of Kleene [Kle51]. A regular expression that correctly identifies valid e-mail addresses following the description above would be:

```
[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}
```

For an explanation of this syntax, the reader is referred to a textbook on the matter (e. g., [Fri06] or [GL12]) or one of the many online resources that describe regular expressions and how they work in detail.

The pattern matching count of a dataset is defined as the number of values that match a specified pattern. Thus, this type of metadata requires a pattern (e. g., in the form of a regular expression) as additional input. For numerous scenarios, pre-compiled patterns exist for validation, like e-mail or IP addresses, website URLs, phone numbers, and currencies. A user can also provide his own pattern in case there are domain-specific requirements on how the data should be formatted.

As powerful as regular expressions are, they also have limitations. Consider for example the International Standard Book Number, or ISBN for short. The ISBN is a numerical code system that is used to uniquely identify each commercially available book worldwide. In its newest version, it states that a valid ISBN consists of 13 digits grouped into 5 parts. While this is easy to check with a regular expression, there is an additional constraint added on top: the check digit. The last digit of an ISBN is the check digit and its value must be the result of a particular calculation involving the other 12 digits. This allows a certain degree of error detection, because small mistakes like mixed up digits result in an ISBN that is most likely invalid instead of a reference to a completely different book. The verification of the check digit is however beyond the capability of regular expressions. Thus, the validation of ISBNs requires special code, which is indeed included in some data profiling tools.

Summary A summary of the various types of metadata that have been described so far in this first part is given in Table 4.3. For every type it lists the name, a short description and a note about restrictions and applicability considerations.

Multiple Fields

This section describes types of metadata that involve more than a single field. These can be multiple columns in a relational database, or different attributes in a document- or graph-oriented data storage. Note that some of the following concepts are usually discussed as constituents of data mining or statistics. The reasons they are included here is that first, they fit the definition of being metadata, i. e., data about data, and second, they are helpful in getting to know and understanding the data in question, which is the main goal of data profiling. Thus, it is reasonable to consider them as metadata types despite their origin.

Correlations. A correlation is a measurement between two sets of data that, broadly speaking, describes how closely they are related to each other [JWHT13, p. 70]. If values from one column can be predicted with reasonable accuracy based on the values of another column, both are said to be correlated. The most prominent type of correlation is a linear relationship, i. e., two sets of data are related such that increasing values in one set imply linearly increasing (or decreasing) values in the other set. This type of correlation is only applicable to numeric data. The underlying cause for this is often a connection between real-world concepts. For example, the height of a person is correlated with his shoe size, i. e., the taller a person is, the larger his shoes are likely to be.

Table 4.3: List of single field metadata types.

Metadata	Description	Note
Name	Label that uniquely identifies an attribute, field or column	Also applicable to collections of data, e. g., tables, graphs, or databases
Row Count	Number of rows in a dataset	Also known as entity count, node count, or instance count
Data Type	Data type of a data value or field, e. g., string or number	-
Null Count	Number of null values	Only applicable within a system that supports null values
Blank Count	Number of blank values, i. e., empty cells	Only applicable to string data types
Default Value Count	Number of values equal to the default value	Only applicable when a default is known
Distinct Count	Number of values that are different from every other value	Also known as cardinality (of a dataset)
Unique Count	Number of values that appear exactly once	-
Duplicate Count	Number of duplicate values	May be defined differently depending on use case
Value Length	Number of characters a value contains	Usually aggregated with `min`, `max` or `avg`
Number of Decimals	Number of decimals in a numeric value	Only applicable for numeric data types. Also known as precision
Value Statistics	Lowest, highest, average or median value in a dataset	Average is only applicable to numeric columns. Median requires an ordered set.
x-th Percentile	Threshold value below which x% of the values are	Only applicable for numeric data types
Mode	Most frequent value, i. e., value that appears most often	-
Value Frequency Table	Table of most (or least) frequent values	Can include absolute and/or relative frequencies
Pattern Frequency Table	Table of most (or least) frequent patterns	Only applicable to string data types
Pattern Matching Count	Number of values that match a specified pattern	Patterns can be provided as regular expressions

There are many ways to measure and quantify the degree of correlation between two datasets. The most common way is to use a correlation coefficient, such as the Pearson correlation coefficient [FMR12]. A correlation coefficient can assume any value between -1 and $+1$, where $+1$ indicates the strongest degree of positive correlation and -1 the strongest degree of negative correlation. A value of zero indicates that there is no correlation in either direction.

In data profiling, correlation is a useful type of metadata because it reveals to a user which part of the data is related to which other part of the data. This can be very helpful in cases where the user is unraveling the semantics behind the data or is looking for a connection between two datasets. For example, a recent study on taxi and weather data has found a negative correlation between the number of taxi trips and the wind speed, i. e., when it is windy, people tend to not take a taxi [CDDF16]. This phenomenon could then be traced back to hurricanes Irene and Sandy.

Additionally, it makes sense to periodically check for correlations in datasets where no correlations is assumed. This is because unexpected correlations have the potential to reveal new and impactful insights. Computing every correlation coefficient in a given dataset requires a comparison of each column with each other column, i. e., the number of steps grows squarely with the number of columns. This makes it computationally expensive and can quickly become prohibitive. Thus, some tools only offer to compute the correlation between columns that the user specifically selects.

Association Rules. An association rule expresses the co-occurrence of two or more values in a dataset. It is written in the form $x \rightarrow y$, which expresses that the value x is likely to be observed together with value y. This is often done in market basket analysis or recommender systems [BMU+97][LRU14, p. 213][SKKR00]. For example, a retailer might find out that the rule `{diapers}` $\rightarrow$ `{beer}` can be derived from his sales data, which indicates that people who bought diapers are also likely to have bought beer. This information can then be used to, e. g., plan promotional offers or optimize shelf arrangements [AIS+93].

Numerous algorithms for generating or *mining* association rules have been developed. One very popular example is the Apriori algorithm, which has been invented by Agrawal and Srikant in 1994 [AS94]. It provides exact results, making it great for smaller datasets, but it is not very scalable. This has prompted development of extensions that use heuristics to sacrifice accuracy results in favor of speed and memory requirements. A detailed survey of algorithms for association rule mining can be found in [HGN00].

One notable characteristic of association rules is that they can be applied to categorical values. This makes them a valid option to choose from if the data at hand is not numeric and other analysis methods are not applicable.

Unique Column Combinations. A unique column combination (UCC) is a set of columns whose projection, i. e., the removal of all other columns, contains only unique value combinations [AGN15]. In other words, a UCC consists of columns that are able to uniquely identify each row of the dataset. For instance, the primary key of a relation is a UCC, because its projection is per definition unique. Note that, despite the name, a UCC can also consist of

just a single column.

Discovering all UCCs in a given dataset is an NP-hard problem [HQRA+13]. Each additional column exponentially raises the number of potential column combinations that need to be examined for uniqueness. This makes it very challenging to construct algorithms that are both efficient and scale to larger datasets. One such algorithm is called *DUCC*, shorthand for *discovery of unique column combinations*, which is presented in [HQRA+13]. DUCC solves the problem by employing depth-first search and random walk strategies to traverse the intermediate graph. An in-depth explanation of this algorithm as well as a comparison with other UCC discovery algorithms are given in [AGN15].

Knowing the UCCs of the data at hand can be helpful in a number of data-related tasks, such as data integration or query optimization. In cases where the primary key of a relation is needed but unknown, UCCs can be used to narrow down the set of possible columns as every UCC is a candidate key.

Inclusion Dependencies. Given two sets of columns A and B, an inclusion dependency (IND) written as $A \subseteq B$ states that each individual value in A also appears in B [AGN15]. For example, in an employee table, it is reasonable to expect that every value in the department column also appears in the department table, i. e., every employee works in a department that is listed as part of the company. Note that INDs are not symmetric and the opposite does not necessarily need to hold. In the example, not every value from the department table needs to appear in the corresponding column in the employee table, which means that there could be departments with no associated employee. In other words, `emp.department` $\subseteq$ `department.name` holds true and is a valid inclusion dependency, but `department.name` $\subseteq$ `emp.department` is false.

INDs have been formalized by Fagin in 1981 [Fag81] and form the basis for referential integrity in relational databases. This is because foreign keys are a specialization of INDs in which the left-hand side is a reference to a primary key on the right-hand side. In other words, INDs are usually the first step when discovering foreign keys. Cases where the left- and right-hand side come from the same relation are called *self-referencing* or *recursive* foreign keys.

An extension of INDs is the concept of a *partial* or *approximate* INDs [LPT02][DLP09]. These are INDs that *almost* hold true, i. e., there is a small number of values that violate the IND, and once removed, turn the partial IND into a proper one. Partial INDs are very useful when dealing with dirty or noisy data, because they can be used for data cleaning purposes.

Another extension is the concept of *conditional* INDs which only hold on a well-defined subset of the data [BAL+12][AGN15]. A well-defined subset in this context can be described by a clear filter rule, such as `department = sales`, which then acts as a condition for the associated IND.

Calculating these different types of INDs is far from trivial. Much research effort has been put into devising and optimizing algorithms that address this challenge. Papenbrock et al. present an efficient and scalable algorithm for exact IND discovery in [PKQRN15], while partial INDs are also considered in [LPT02].

Table 4.4: CloudHost customers dataset.

Customer ID	Zip code	City
1	48143	Münster
2	48565	Steinfurt
3	48149	Münster
4	48565	Steinfurt

Functional Dependencies. A functional dependency (FD) is a relation between two sets of columns in a relation that states that any two rows that have the same values in the first set of columns must also have the same values in the second set of columns. Formally, this can be written as follows: "A functional dependency over R is an expression of the form $X \rightarrow A$, indicating that $\forall r_i, r_j \in r$ if $r_i[X] = r_j[X]$; then $r_i[A] = r_j[A]$." [AGN15, p. 570]. To illustrate this concept, consider Table 4.4, which shows a small excerpt of CloudHost's customer data, with customer IDs, zip codes and cities. It can be observed that each distinct zip code is associated with the same city, which translates into the FD `zip code` $\rightarrow$ `city`. In other words, it can be said that the zip code *functionally determines* the city. This FD would be violated if a row were entered that contained, e. g., 48143 as zip code and any other string than 'Münster' as city.

FDs can be found in every relation. When a primary key is set, it functionally determines every other column. In the example case, the FD `customer id` $\rightarrow$ `zip code, city` holds. FDs based on primary keys are usually not very useful, because primary keys are by definition unique, and unique columns, i. e., columns with only unique values, always functionally determine all other columns. Furthermore, for any given set of columns A, the FD $A \rightarrow A$ holds. However, not much can be learned from such an obvious FD, which is why they are called *trivial*. More generally, every FD $A \rightarrow B$ is called *trivial* when B is a subset of A.

More useful are FDs that are neither based on a unique column nor trivial. The FD `zip code` $\rightarrow$ `city` for example can be used to impute missing values: Assume that a new row is entered which has 48565 as zip code but null as city. This missing value poses a data quality issue and should be resolved as soon as possible. Domain knowledge shows that `zip code` $\rightarrow$ `city` holds, and thus, the missing city value can be inferred from the zip code. This is done by looking up 48565 in the other rows and taking the city value 'Steinfurt' from there. It does not matter whether row 2 or 4 is found first, because they both have the same city value. Note that this obviously requires that the zip code in question is already present in the data or can be looked up from another source. This concept is similar to the chase algorithm, which was introduced by Aho et al. [ABU79].

Further application areas for FDs can be found in the normalization of databases where they are used to search for violations of the second and third normal form. Another area is data warehousing that uses a denormalized star schema. Here, FDs can be used to gain insight into potentially hidden dimension hierarchies. Generally speaking, the presence of an FD reveals some of the semantics of the underlying data, because it shows how the individual parts are related and connected. This makes them a very useful concept in any data profiling scenario.

Table 4.5: List of multi field metadata types.

Metadata	Description	Note
Correlations	Measures the degree to which two datasets are related	Usually quantified using correlation coefficients that range from -1 to $+1$, e. g., the Pearson correlation coefficient
Association Rules	Expresses co-occurrences of values	Applicable to categorical values
Unique Column Combinations (UCC)	Set of columns that uniquely identify each row	Useful when searching for primary key candidates
Inclusion Dependencies (IND)	Relation between two sets of columns that states that each value on one side must also appear on the other side	Prerequisite for foreign keys and referential integrity. Extensions include partial and conditional INDs
Functional Dependencies (FD)	Relation between two sets of columns that states that values on one side functionally determine values on the other side	Used for database normalization. Extensions include partial and conditional FDs

Similarly to INDs, there are variants of FDs that are *partial* or *approximate*. The intuition here is that these are FDs that *almost* hold, i. e., there are violations, but their number is small with respect to the overall number of rows in the relation. Another FD variant are *conditional* FDs. These holds under a specific condition, which specifies a subset of the rows that are considered. For example, if the customer dataset were to be extended worldwide, but the FD only holds for Germany, this could be expressed as conditional FD with the condition `country = 'Germany'`.

One of the main problems with FDs is their efficient discovery, because the straight-forward enumeration of all possibilities is prohibitively expensive due to exponential scaling. This mandates the design of clever algorithms that exploit certain characteristics of the search space to achieve optimization. A comprehensive overview and experimental evaluation of FD discovery algorithms can be found in [PEM+15].

Summary A summary of the various types of metadata types that have been described in this second part is given in Table 4.5. For every type it lists the name, the abbreviation that is used as subscript in the formal notation, a short description and a note about restrictions and applicability considerations.

Multiple Datasets

From a practical perspective, it might be argued that there is a third class of metadata types that encapsulate information that is spanning multiple datasets, e. g., multiple tables. A prime example here is the concept of foreign keys, which link together tables in a parent/child relationship. To identify a foreign key relationship, both tables need to be considered simultaneously.

Upon closer examination however, it becomes apparent that these metadata types spanning multiple datasets are just a special case of multi-field metadata as described in the previous subsection. Any of those metadata types are also applicable if the involved fields are spread across multiple datasets. In particular, foreign keys are a subclass of inclusion dependencies.

To conclude, there is effectively no "third class" of metadata types, because the distinction into single field and multiple fields metadata types is already all-encompassing.

4.2.2 Classifying Types of Metadata

To operationalize metadata and further promote their research, it is useful to provide a classification for them. This is however not an easy task, because the variety of metadata is as diverse as the data that is described. There are numerous disciplines and domains that make heavy use of the concept of metadata, such as statistics, information systems, databases, health care, law, libraries, or media file formats for music, video and photography. Each of these areas has its own requirements and thus, there is no unifying framework, model or theory that encompasses everything. The National Information Standards Organization (NISO) published what they call a "comprehensive overview of metadata, covering topics such as metadata types, standardization and use" in January 2017 [Ril17]. However, this overview focuses primarily on the cultural heritage world and the usage of metadata in bibliographic and library applications. Thus, it is rather narrow in scope and does not provide much insight for the information systems point of view that this thesis aims to take.

More can be learned from the mature research area of data warehousing. One of the pioneers of data warehousing is Bill Inmon, who declares that "one of the most basic ways to divide the world of metadata is by technical metadata versus business metadata." [IOF08, p. 16]. Technical metadata is metadata that is useful for a database technician to do his job, especially "design, development, maintenance, and other functions" [IOF08, p. 12]. Inmon et al. give the following examples for technical metadata:

- Database table name
- Database index name
- Database table layout
- Database field name
- Field physical characteristic
- Field constraints

- Inter-record or intertable relationship

Simply put, technical metadata is about what is "under the hood" to make things work as expected. Note that Inmon et al. only consider the relational data model by explicitly using the term *table*.

Business metadata on the other hand includes those types of metadata that are "useful to the business-person in the day-to-day conduct of business" [IOF08, p. 12]. A business user does not care much about how tables and fields are named, or what data types they have. He is more interested in the content, and thus, needs business metadata that helps him find and interpret the data.

Another famous data warehouse expert is Ralph Kimball, who also acknowledges business metadata and technical metadata as the two main categories. Additionally, he also introduces a third category, called process metadata, which is used to capture the result from various processes and operations that surround the data, such as ETL processes or queries [KRB+16, p. 710]. However, this category does no longer describe the underlying data, and thus, does not quite fit the notion of "data about data". Instead, it is data about the processes that deal with the data. While this type of operational data is without a doubt important for the operation of a data-driven system, there is no immediate applicability during the data profiling and data understanding phase. Thus, it is not further investigated in this thesis.

More detailed work on the matter was conducted by Abedjan et al. in [AGN15]. They describe a classification of what they refer to as "data profiling tasks" (see Figure 4.2), which are individual tasks performed in a data profiling scenario. In their work, Abedjan et al. regard data profiling from an algorithmic point of view and arrange and group the individual tasks accordingly. This differs from the more user-centric view point taken in this thesis. However, each of the tasks describes a class of computations that result in a specific type of metadata. Thus, there is almost a one-to-one correspondence between these tasks and the resulting metadata, which makes it very useful in the context of metadata classification.

At the highest level, Abedjan et al. distinguish three classes: single column, multiple columns, and dependencies. The single column class comprises all those tasks that look at only one isolated column at a time, which is similar to the descriptions in the first part of Section 4.2.1. The following four sub-classes are identified:

Cardinalities are the result of basic count operations. This includes counts such as the number of rows, nulls, duplicates or uniques, as well as aggregations of value lengths, e. g., the minimum, maximum, or average value in a column. Apart from calculating the exact results for these values, there also methods to *estimate* them, which also fall under this category.

Patterns & data types take a more in-depth look into the actual values of the data. Patterns are extracted by replacing characters with more generic representations. Subsequent aggregation can then reveal the distribution of patterns across a column. The data type on the other hand is a classification that determines the domain of permissible values. Common data types are integers, character strings, Boolean values, or decimal numbers. From these basic types, more complex data types can be constructed. For example, to store information about dates, the data type *date* could be defined by enforcing the

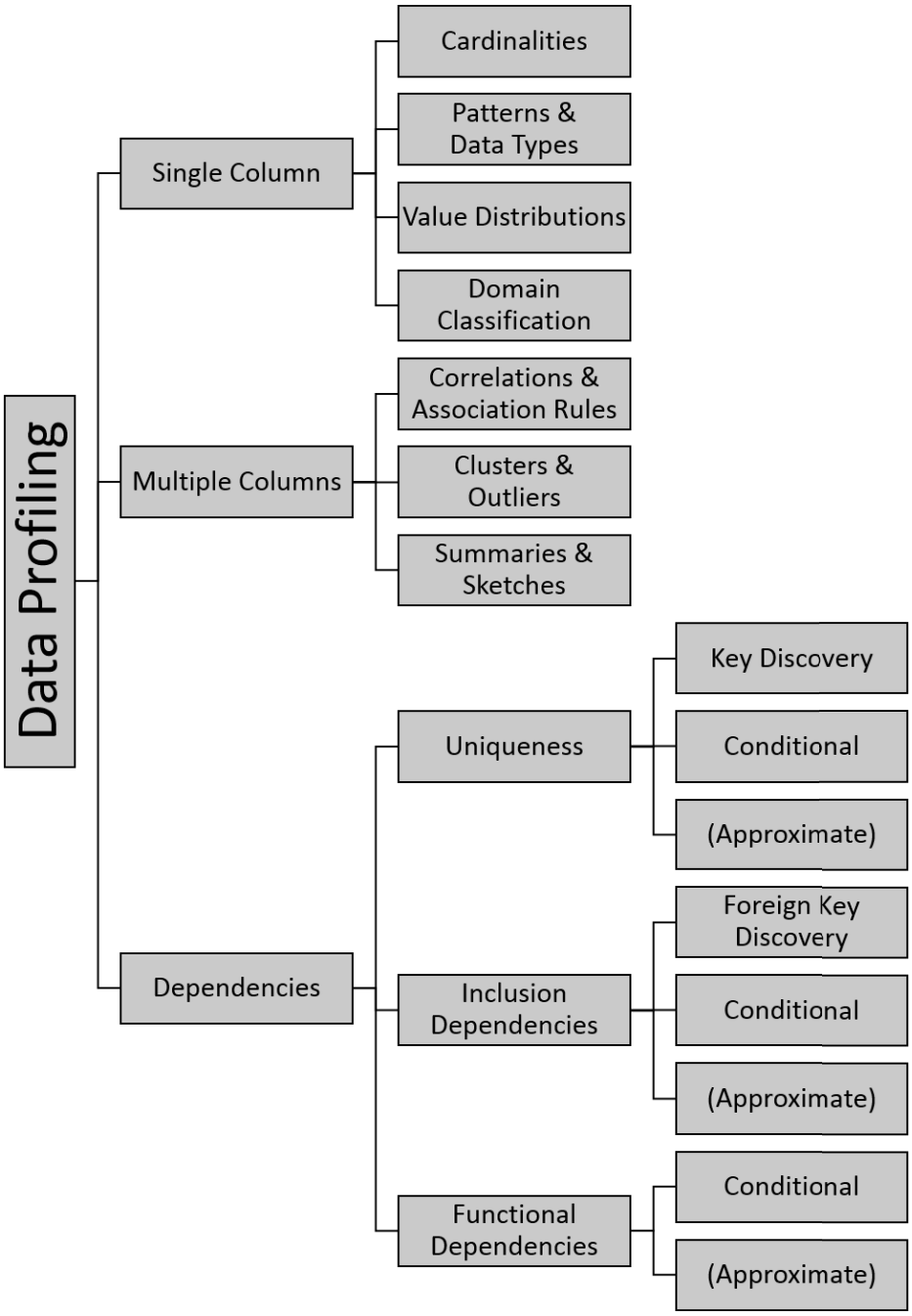

Figure 4.2: Classification of data profiling tasks according to Abedjan et al. Source: [AGN15].

pattern `YYYY-MM-DD`. This data type uses a combination of integers and the character symbol - to precisely specify how a date should be represented. During profiling, this usually works the other way around, e. g., when a pattern of the form `dddd-dd-dd` is frequently encountered, it is reasonable to assume that these values represent a date. Thus, patterns and data types are closely related and grouped together here.

Value distributions describe how the values within a column are distributed along their domain. The simplest form is a table that shows the most (or least) frequent values alongside their total number of occurrences. More sophisticated techniques within this sub-class include histograms, box plots, and graphs of distribution functions.

Domain Classification is focused on the semantics that are represented by the data. Frequently encountered domains are for example *first name* or *city*. Deducing the semantic domain of a column usually requires external business knowledge or sophisticated ontologies and is thus very hard to automate.

The next class considers multiple columns at once to extract insights that are spread across the dataset. Abedjan et al. describe the following three sub-classes:

Correlation & Association Rules have in common that they both aim to quantify the relation between two columns. One key difference is that correlation is usually applied to numerical data, whereas association rules are useful when working with categorical data.

Clusters & Outliers are opposing concepts that assess the homogeneity of a group of objects. Clustering aims to group objects together that are similar in some sense [JWHT13, p. 373]. This requires a quantification of similarity, which is often done by using a distance function (e. g., Euclidean distance). Such a distance function is often set up as a combination of multiple columns. For example, people could be grouped together if they have similar age, body weight and height. The result is an additional attribute that is attached to each object in the group that shows to which cluster it belongs. Outliers on the other hand are those objects that do not fit particularly well in any cluster, i. e., their attribute values deviate greatly from the others [JWHT13, p. 96].

Summaries & Sketches are a general group of methods that aim to find a brief and compact description of a given dataset. They make use of previously described techniques such as clustering and association rule mining to trim down the amount of data that is displayed. The quality of such approaches can be measured with the two metrics *compaction gain* and *information loss* [CK07].

The last class in the classification scheme by Abedjan et al. is called *dependencies*. It is worth mentioning that technically, dependencies are also multi-column concepts. The authors declare that "[while] dependency detection falls under multi-column profiling, [they] chose to assign a separate profiling class to this large, complex, and important set of tasks." [AGN15, p. 560]. This once again highlights the algorithmic point of view they take, because this distinction is based on technicalities rather than intrinsic characteristics of the underlying concepts and how the user perceives them.

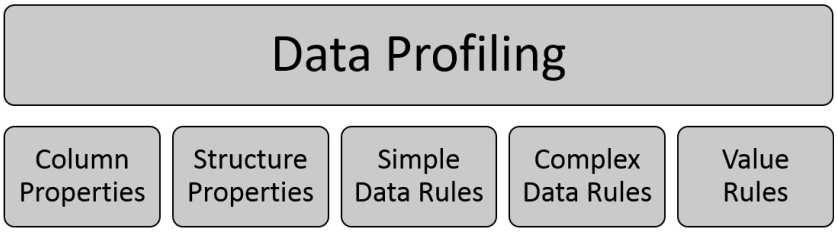

Figure 4.3: Data profiling outputs according to Olson. Source: [Ols03].

The dependencies class consists of three sub-groups, namely uniqueness, inclusion dependencies, and functional dependencies. All these three concepts have been introduced and described in the second part of Section 4.2.1.

Further tasks are not considered by Abedjan et al. to be part of data profiling. Thus, some types of metadata and structural properties of data are not covered by this classification. In particular, the special relationships generalization/specialization and is-part-of/is-component-of are not mentioned. Further, a purely data-centric view is adopted in which only metadata that can be derived from the data itself is considered. This leaves out those types of metadata that are external to the data. These will be dealt with in Chapter 6.

A different classification scheme is introduced by Olson, who discusses what he considers to be the five outputs of data profiling. These five outputs are shown in Figure 4.3 and are detailed next.

Column Properties are properties that are derived from inspecting single columns individually. For example, Olson lists the name, data type, character set and length restrictions as typical column properties, among others [Ols03, p. 149].

Structure Properties consider how the data is structured across its columns. In this group fall the following: functional dependencies, primary keys, foreign keys, normal forms, and synonyms. Apart from the last one, they have all been described already. Synonyms are a concept that applies when two or more columns contain the same business facts [Ols03, p. 184].

Simple Data Rules are data rules that concern a single row or entity. Data rules in this context are "specific statements that define conditions that should be true all of the time" [Ols03, p. 215]. These conditions are defined over multiple columns and specify admissible value ranges in one or more columns based on the values in one or more other columns. For example, the simple data rule `IF EMPLOYEE.STATUS = 'PART_TIME' THEN EMPLOYEE.DEPARTMENT != 'RESEARCH'` states that any part-time employee may not be assigned to the research department. Once they are explicitly known, data rules can be implemented and enforced in databases using triggers.

Complex Data Rules are data rules that concern a set of rows or entities. For example, a complex data rule could specify that the order quantity for any part must be at least as high as the minimum order quantity defined in the master data.

Value Rules are rules that are derived from the definition of a value computation [Ols03, p. 246]. For example, the frequency distribution of a column involves the computation of distinct values and counting their occurrences. According to Olson, this constitutes a value rule. Other types of value rules are extreme values or group aggregations.

There are some interesting points that can be learned from comparing Olson's classification with the previous one from Abedjan et al.. First, column properties are very similar to the single column-group. Next, the structure properties largely overlap with the multi column-group, with the notable exception of *synonyms*. To reconcile these differences, it needs to be understood what synonyms in the context of databases are. Olson explains that there are four types of synonyms:

1. Primary key/foreign key synonyms, which occur in every primary key/foreign key pair, because they are defined to contain the same business facts.

2. Redundant data synonyms, which occur when columns are duplicated across tables. These are special cases of a functional dependency and can be eliminated without information loss.

3. Domain synonyms, which have no structural relation, but just happen to contain the same business fact. For example, a 'city' column may appear in the 'user' table as well as in the 'employee' table. This phenomenon should be registered as a (partial) inclusion dependency.

4. Merge synonyms, which are essentially schema matchings, i. e., correspondences between different data sources that state which columns contain the same data and should be merged into the same target column. This plays an essential role in any data integration scenario.

To conclude, although the name *synonyms* is never used by Abedjan et al., the concept can be derived from the established dependencies and the differences are purely down to naming conventions.

A more notable difference is the concept of *data rules*, which are not considered by Abedjan et al., while they play an important part in Olson's classification. The reason for this is that Abedjan et al. is focused on metadata that can be derived directly from the data without any other input. This is in stark contrast to Olson's approach, who considers documentations, logs, domain knowledge and other external sources as valid input for the profiling process. Data rules in particular cannot be discovered or generated from the data in the same way that it is possible for, e. g., functional or inclusion dependencies. Olson writes: "The potential for data rules is almost infinite from a discovery point of view. It would not be practical or beneficial to discover data rules from the data" [Ols03, p. 220]. Instead of discovering them, other methods for establishing of data rules are proposed: source code scavenging, analysis of database-stored or business procedures, or simply speculation.

Lastly, what Olson calls *value rules* is already part of the single column-group of metadata types described by Abedjan et al..

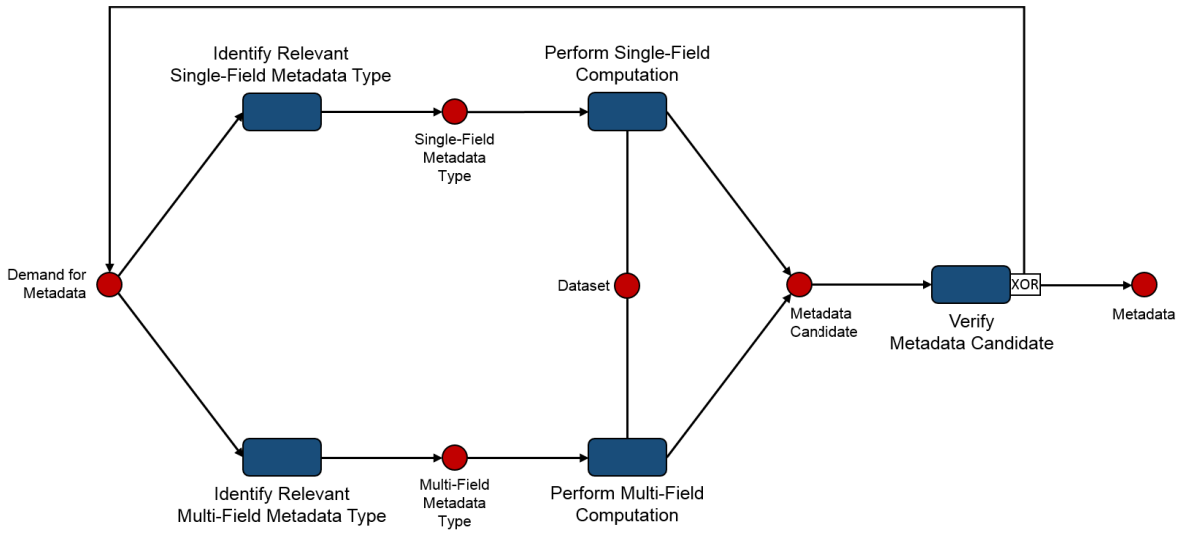

Figure 4.4: Proposed process for the extraction of metadata.

4.2.3 Metadata Extraction Process

In order to operationalize the extraction of metadata, it makes sense to conceptualize it as a process, as shown in Figure 4.4.

The process starts on the left-hand side with an arbitrary demand for metadata. This demand comes from the individual use case at hand. For example, when a new dataset is acquired, or an existing dataset has undergone significant change, a demand for metadata can be identified. This demand is further broken down into the two classes of metadata types, single-field and multi-field. For each class, the relevant types of metadata are identified by matching the available types (cf. Tables 4.3 and 4.5) with the respective demand. This results in a set of metadata types, whose values are computed in the next steps. In order to carry out this computation, the dataset in question is used in a read-only fashion, i. e., it remains unchanged. All results are gathered as metadata candidates, which undergo a final verification step. This verification is put in place to make sure that the results match the initial demand. If that is not the case, the demand is refined and fed back to the start of the process, such that a new iteration can begin. This can be necessary, if, for example, the calculated metadata is insufficient and further metadata types are required. If the verification step succeeds, the resulting metadata is put out.

4.3 Three Challenges of Data Profiling

This section presents a different way to think about and structure data profiling activities. Whereas the previous section focused on the metadata as the result of profiling, the following remarks apply the input-process-output (IPO) model to data profiling. The IPO model has a long history in computer science, particularly in software engineering and programming [Gra10, p. 165][Goe10, p. 11][CFH05, p. 99]. It decomposes any system or function into the three components *input, process* and *output*. Landon and Landon describe these three

components as the "basic activities" of any information system [LL95, p. 7]. The *input* is a description of what the input is, how it should be organized or formatted, and which requirements it needs to fulfill. For example, when programming a function, this corresponds to the parameters of that function, which can be specified to be of a particular data type, e. g., integer. Next, the *process* describes what is being done with the input. This corresponds to the code in the body of the function and specifies exactly which steps are done in which order. Lastly, the *output* is a specification of how the result looks like and which requirements it should fulfill. In programming, this corresponds to the return type, which states the data type of the variable that is being returned by the function.

Abedjan et al. also apply the IPO model and define three core challenges of data profiling based on the former. These are managing the input, performing the computation, and managing the output [AGN15] and are discussed more extensively in the following subsections.

4.3.1 Managing the Input

The input for data profiling is a dataset, as introduced in Section 3.1. There are several issues that need to be addressed in this stage. First, the dataset needs to be specified, i. e., its location and access method must be made available. This can range from pointing towards local files to configuring firewalls for remote access to online sources. Once the dataset is accessible, the part of the data that should be analyzed needs to be specified. In most cases, only a subset is of interest, e. g., a specific group of tables or columns, but analyzing the complete dataset at once is also possible.

The next task is to look for potential data formatting and parsing issues, and to resolve them. If the data resides in a CSV file, it needs to be specified what delimiter should be used to load the data. Despite the name, it is often the case that the comma character is not the delimiter. Other characters are permitted, with the tab and semicolon being popular choices. Surprisingly, many tools that load CSV files do not have a feature to automatically guess the most likely delimiter in a given file, so this task is usually done manually. Other formatting issues that need resolving are the configuration of the correct encoding, text qualifiers and escape sequences. Note that most of these tasks can be omitted when a proper DBMS is used as a source.

When the data is correctly parsed, the user needs to specify which metadata he wishes to have computed. Ideally, the data profiling tool assists the user by showing him which types of metadata are applicable to which parts of the data. Alternatively, the user can also choose to simply calculate everything and then search through the results to look for interesting insights. This approach is obviously only feasible in cases where the resulting metadata are quickly calculated and easy to review.

After this challenge is addressed, i. e., when the input has been managed, the tool can commence with performing the computations.

4.3.2 Performing the Computations

This challenge has received the most attention in data profiling research [AGN15, p. 557], and deals with algorithms for metadata extraction, their runtime, and optimization. Due to

the ever-increasing volumes of data, it is important that any algorithm that processes data is efficient and scalable. The challenge here is to cope with datasets that potentially have not only many rows, but also many columns. As mentioned before, the calculation of some metadata types scales exponentially with the size of columns, e. g., the discovery of inclusion or functional dependencies.

One way to deal with this is to apply the KIWI principle, which stands for "kill it with iron", i. e., throw as much computational power at the problem until it is solved in a reasonable amount of time. This is especially effective if the problem is structured in such a way that it can be addressed through an algorithmic approach that can benefit from parallelization.

Alternatively, or in combination, heuristics can be applied. A heuristic is any method that provides a practical approach to a problem without guaranteeing optimal results, only "reasonably good" ones [Pea84, p. 3]. Thus, a heuristic sacrifices accuracy in favor of reducing the runtime complexity. Ideally, this trade-off can be measured or estimated in some way, so that the loss of accuracy can be assessed. In many cases, approximate results are good enough, making heuristics an enticing solution. This is especially true in data profiling, because most of the results are not used as the basis for further computations, but instead are manually inspected and assessed by a user. The goal often is to "get a feel" for the data, which is easy enough to achieve with approximate results.

Performing the computation is done automatically by a computer or system. This means that some users may be tempted to treat it like a black box where they put the data in and get metadata out, and to not care about the details that take place in between. However, this is a dangerous practice, because ignorance about how a piece of metadata has been computed impedes one's ability to correctly evaluate and interpret it. Thus, it is strongly recommended that anybody who uses data profiling tools and attempts to interpret their results makes himself familiar with the metadata types and their definitions beforehand.

4.3.3 Interpreting the Output

In the last challenge, the user is presented with the results. The task of making use of these results by interpreting them in their respective context cannot be automated by a computer. Abedjan et al. state that "profiling results need interpretation, which is usually performed by database and domain experts" [AGN15]. Furthermore, there is no standardized reference process that guides the user in this task. As such, it is the hardest of the three challenges and little work exists that addresses it.

One of the reasons for this is that the interpretation is technically no longer part of the profiling process itself. Usually, profiling is done to learn something about the data that is useful in some other, broader context. Thus, the interpreter needs to consider this broader context because it provides the direction in which the user should focus his attention. However, the broader context is dependent on the use case and the profiling context and cannot be easily generalized.

It is an open research challenge to figure out what can be done about this. One observation is that seasoned data experts are confident in their ability to interpret metadata. This is because they have done it before and use their experience that tells them what to look for, what to expect, and what is unusual in a given data profile. Thus, practice makes perfect, as

the expression goes, and one needs to interpret data profiles to become better at interpreting data profiles. This idea could be supplemented by a compilation of best practices and practical examples.

4.4 Software Tools and Research Projects

Data profiling and metadata extraction can, to a certain extent, be performed in an ad-hoc fashion by hand-writing queries and executing them, e. g., using SQL as demonstrated above. However, the task of manually writing queries is prone to errors and can quickly become cumbersome as the complexity of the data increases. Furthermore, many metadata types require more involved computations for which query languages are inappropriate. This is why numerous dedicated data profiling software tools have been implemented, which support a user by offering easy access to profiling algorithms and functionalities. The same train of thought is followed by Kimball, who writes "you can be much more productive in the data profiling stages of a project using a tool rather than hand coding all the data content questions." [KR13, p. 450].

This section has two goals: First, it aims to give an overview of the various classes of software tools available for data profiling. Second, a more detailed impression of some of the tools and their usage is provided by describing selected representatives. The classification scheme suggested here distinguishes between three classes of data profiling tools:

- Dedicated Tools
- Integrated Tools
- Research Projects

Dedicated tools are stand-alone programs that are built with the main purpose of providing data profiling or similar functionality. These are described in Section 4.4.1. Integrated tools on the other hand are not offered stand-alone, but instead are part of a bigger package, usually a data management platform with many features. They are described in Section 4.4.2. The third class, research projects, contains contributions from (computer) scientists to the data profiling discipline. As these research projects tend to focus on specific techniques or algorithms, their range of features as well as their usability is usually significantly lower than tools from the first two classes. Additionally, these projects tend to be abandoned more quickly, because there is little commercial interest to maintain them. Section 4.4.3 provides some examples.

4.4.1 Dedicated Tools

A selection of popular dedicated data profiling tools was compiled by querying the search engines Google, Bing, startpage.com, and DuckDuckGo. Using the search term "data profiling tool", the top-two results returned by all engines were consistently *Talend* and *Datamartist*. Thus, it is concluded that there is at least some degree of popularity and relevance to these two tools, and they shall be examined more closely to demonstrate data profiling in practice.

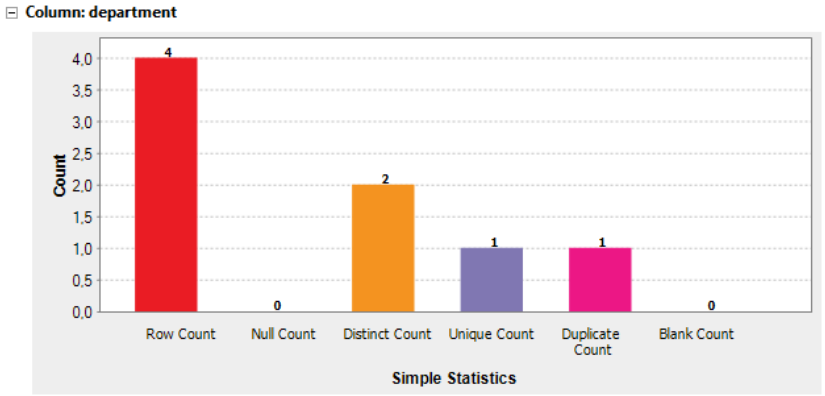

Figure 4.5: Column analysis for the department column using TOS DQ.

Talend Open Studio for Data Quality. Talend Open Studio for Data Quality (abbr. TOS DQ) is an open source tool that offers a wide array of data profiling functionalities [34]. It consists of a graphical user interface and supports data sources that either reside in a supported database or in a flat-file format like CSV. Among the supported databases are well-established relational systems such as IBM DB2, Microsoft SQL Server, MySQL, Oracle Database and a general JDBC interface. Support for modern Big Data systems on the other hand is more sparse, with Apache Hive and Impala being the only notable entries. Overall, twenty different database systems are supported.

Once the data is loaded into the tool, the user can choose from a variety of analyses. These range from general information about the database as a whole (e. g., number of tables, rows, views, keys and indexes) to metadata extraction on specific columns, with many of the metadata types described in Section 4.2 being supported. However, the more complicated types that involve multiple columns (e. g., inclusion dependencies or unique column combinations) are not supported out of the box, and while there is a function for functional dependency analysis, it works in a rather minimalistic way. The user has to manually specify the determinant and dependent columns between which a dependency is assumed, and the tool will calculate a dependency strength for that pair. Thus, there is no automatic discovery of functional dependencies.

Additionally, TOS DQ allows users to define their own types using SQL templates or Java code. These user-defined types can be used to, e. g., calculate and monitor business- or domain-specific metrics that measure data quality or other concepts.

One of the core features of TOS DQ is column analysis, which allows the extraction of metadata for individual columns. Note that Talend uses the term *indicators* to refer to what has been established as *metadata types* here. When the user has selected the column(s) he wishes to analyze and metadata types for each column, the computation can be started. Afterwards, the results are visualized using suitable charts. For example, back in the example dataset introduced in Table 4.1, the column 'department' contained four values: Sales, HR, HR, and HR. This column is analyzed with the following metadata types selected: row count, null count, distinct count, unique count, duplicate count, and blank count. The result is shown in Figure 4.5.

Column Name	Data Type	Null Rows	Missing Rows	Populated Rows	Completeness	Cardinality	Uniqueness	Min	Max	Average
id	Number	0	0	4	100%	4	100%	1	4	2,5
name	Text	0	1	3	75%	4	100%	--	--	--
age	Number	0	1	3	75%	4	100%	38	45	41,66666...
department	Text	0	0	4	100%	2	50%	--	--	--
hours	Number	0	0	4	100%	3	75%	19,875	40	34,84375

Figure 4.6: Default profile of the employee dataset using Datamartist.

TOS DQ is developed and maintained by Talend Inc. from California, United States. Initially, TOS DQ was called "Talend Open Profiler", which hints at the focus the developers originally had. In 2011 it was renamed to its current name Talend Open Studio for Data Quality, which was most likely a consequence of the much larger popularity and wide-spread use of the term "data quality" in practice.

Datamartist. Datamartist is advertised as a "flexible, visual, data profiling and data transformation tool" [4]. The transformation part refers to its capabilities as an ETL tool which are of little interest here. Profiling is done similarly to TOS DQ. First, a data source needs to be set up, which can be a text or Excel file, or a database. The list of supported database systems contains only seven entries, which include well-established ones such as MySQL, Oracle DB and PostgreSQL. Newer database systems or Big Data sources are not supported.

After the data has been loaded and without any further configuration, Datamartist computes a default profile. This default profile contains some basic metadata as depicted in Figure 4.6, where the employee dataset has been loaded. In this view, the user can drill down and interactively browse through the data and, e. g., find the row with the missing data with a few clicks. Depending on what is currently displayed, the view changes dynamically between a tabular layout and appropriate visualizations such as bar charts.

Next to the default profile, which is automatically computed for every data source, Datamartist offers a designated *data profiler block*. Note that *block* is part of the tool's terminology and denotes a component that performs an operation. The data profiler block has three outputs called 'columns', 'values', and 'formats'. 'Columns' is identical to the aforementioned default profile and lists metadata per column. 'Values' gives a tabular overview of each distinct value and how often it occurs in each column, while 'formats' is just used as a synonym for patterns, i. e., it generalizes values through replacement and then counts their occurrences Both of the latter concepts have been introduced in Section 4.2.1, where they were named 'value frequency table' and 'pattern frequency table', respectively. More advanced functionality, e. g., for dealing with multi-column metadata, is not provided.

There are numerous more dedicated data profiling tools available on the market that cannot possibly all be described here. Some of the names of such tools that have been on the shortlist for consideration here are (in no particular order): Ataccama DQ Analyzer, Experian Data Quality, Trifacta Wrangler, Informatica Data Explorer, and IBM InfoSphere Information Analyzer. Their main commonalities are that they all enable a user to quickly extract most of the single field metadata types described in the beginning of this chapter. The differences between these tools lie in their capabilities of visualization and data ingestion, i. e., which

types of data sources are supported.

4.4.2 Integrated Tools

Many software companies bundle their products into monolithic platforms that cater to a wide range of audiences. In the area of data management, this can make a lot of sense, because it frequently occurs that a user needs access to a wide range of functionalities while working on a particular dataset. For example, during data integration, a user may want to profile the data sources first, then pre-process and clean them in some way, model a target data schema, perform schema mapping, define an ETL process, and finally perform and monitor the integration. Having one single software tool that supports all these individual steps in one interface can greatly facilitate this task.

The data profiling capabilities of these tools can vary widely and are not always clearly marked as such. The metadata of the processed dataset is often integrated in places where the developer thought it might be helpful. For example, any spreadsheet tool, e. g., Microsoft Excel, offers a convenient view of the number of rows in a given selection. This is a very low-level form of integrated profiling, because it displays metadata to the user to help his understanding. However, in some cases this can already be sufficient.

Most major software companies offer a monolithic data management platform that has some form of data profiling capabilities. For example, Oracle has the *Oracle Data Integrator*, SAP has the *SAP Information Steward*, and Microsoft has the *Microsoft SQL Server Management Studio*. All of these tools are sold commercially and targeted at business users, which makes getting access to them rather difficult. However, assessing and comparing these tools is not the goal here, so this section remains deliberately short compared to the other sections.

4.4.3 Research Projects

Metanome is the name of a data profiling platform that was developed as a cooperation between the Hasso-Plattner-Institut in Potsdam, Germany, and the Qatar Computing Research Institute. It is designed with extensibility in mind, which means that users can easily implement their own algorithms and plug them in to test and evaluate them. An overview of the architecture is shown in Figure 4.7. At its core is the backend, which handles algorithm execution either locally or remotely. The configuration and result presentation is done through a Web-based frontend that is compatible with any ordinary browser. A number of input source connectors for popular databases come prepackaged with Metanome, including MySQL, DB2 and Oracle, and also various flat-file formats.

One of the design goals of the Metanome project was to create a platform for algorithm comparison and subsequent optimization. Naturally, the need for optimization is only given for metadata types whose calculation is computationally expensive due to their inherent complexity. Four metadata types in particular are the focus of Metanome. These are UCCs, INDs, FDs, and cardinality estimation (not pictured). For each of these, multiple algorithms can be downloaded from the Metanome store [21], and their respective Java source code is also freely available from a GitHub repository [25].

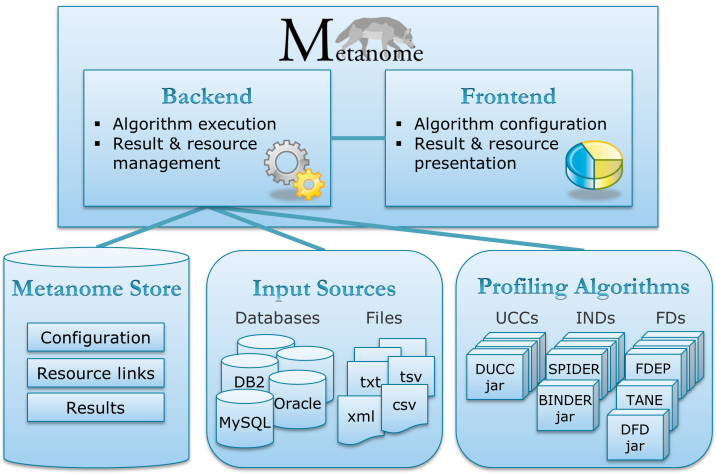

Figure 4.7: Architecture of the Metanome profiling platform. Source: [PBF+15].

Metanome offers no easy-to-use feature for the extraction and visualization of less complex metadata types. As such, it is not well suited for smaller profiling tasks of everyday users, but rather a tool for researchers and developers to implement and test new algorithms.

Bellman Data Quality Browser has been developed by the research team of AT&T labs in 2002. While not explicitly called a tool for data profiling, the authors describe it as a tool for "data mining on the *structure* of the database" [DJMS02]. It offers features to find resemblances within sets, multisets and substrings, which are useful to quickly assess the similarity of different data sources. Further features include the identification of (minimal) keys, join paths, composite fields, and heterogeneous tables.

A join path is a description of how a given table can be joined with another table. Composite fields are the result of the combination of multiple fields, e. g., through concatenation, and thus bear a close resemblance to the concept of synonyms. Candidates for composite fields are produced by applying q-gram signatures [GIJ+01]. Lastly, heterogeneous tables are described as tables that have been altered since their initial creation trough, e. g., the addition of columns. Such a change can potentially cause the affected table to be compatible to only a subset of related tables. The identification of heterogeneous tables is thus a problem that is related to the discovery of join paths.

Overall, the authors claim that the "results of the database structure mining allow the analyst to make sense of the database content" [DJMS02]. This aligns perfectly with the goals of data profiling, and thus, leads to the conclusion that the Bellman Data Quality Browser is essentially a data profiling tool. Furthermore, it has been demonstrated that Bellman can be used to track changes in data over time to uncover patterns of modifications and document the evolution of a database [DJM06]. However, it seems that further development of this tool has been halted, because there have been no updates to it since 2006.

There is a number of further research projects that have resulted in usable data profiling tools, including in no particular order Potter's Wheel [RH01], Data Auditor [GKKS10],

RuleMiner [CIPY14], MADLib Analytics Library [HLK+12] and ProLOD++ [AGJN14]. A short survey that describes some of these tools is given in [AGN15].

5 Data Profiling Variants

In day-to-day business, data profiling can be performed in a number of different ways by a number of different people. Although the core is always about data and the extraction of associated metadata, there exists a variety of different profiling manifestations that can be distinguished. To provide a foundation for further academic discourse and distinguish more precisely between the various kinds of data profiling, this chapter provides a characterization of various profiling approaches. Additionally, this chapter gives more detailed insights into how the previously described profiling techniques and metadata types can be put to use in a practical setting.

The characterization that is presented here consists of the following seven dimensions: purpose, input, executor, method, output, user, and application areas. These dimensions were derived from the IPO model (cf. Section 4.3) and focus additionally on the people involved, i. e., the executor and user, and the areas in which a profiling variant can be applied. Each dimension addresses a specific question, as shown in Figure 5.1.

Each section in this chapter describes one data profiling variant and its properties with respect to the dimensions. The following variants are considered here:

- Data-oriented data profiling
- Goal-oriented data profiling
- Multi-source data profiling
- Data profile validation
- Visual data exploration
- Data discovery

Note that these variants are not intended to be mutually exclusive, i. e., they can overlap or intersect so that a single profiling endeavor may fall into multiple of the proposed categories. This allows for more flexibility when applying these categories to a specific use case at hand. Additionally, this list does not claim exhaustiveness. Depending on the granularity and objective, it could be further subdivided or complemented with fringe activities. The aim here is to give an overview over the most commonly observed data profiling variants and explain how they can be characterized and applied.

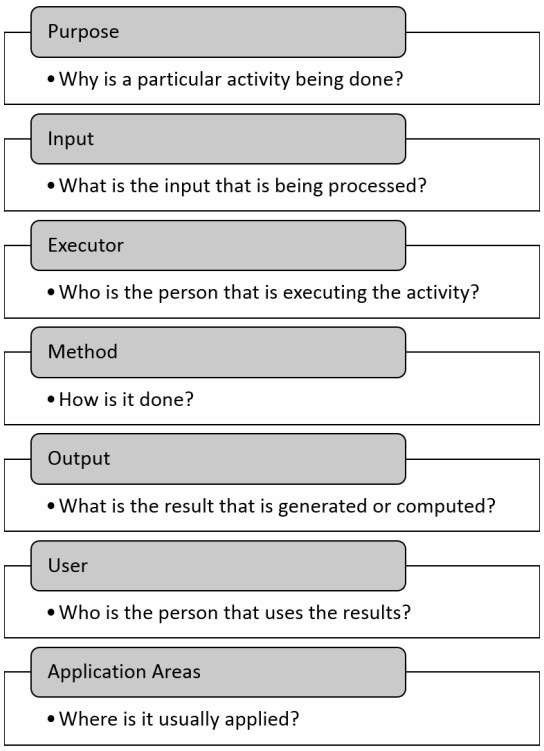

Figure 5.1: Characterization dimensions for data profiling variants.

Figure 5.2: Data-oriented data profiling.

5.1 Data-Oriented Data Profiling

This is the canonical, most basic setting, which is usually meant when people refer to *data profiling* without additional discriminator. A single data source is the input for which a data profile is being created. It is called *data-oriented* to indicate that the input data is the focal point of this variant, and to differentiate it from another type of orientation, *goal-oriented*, which will be described in the next section. Naumann refers to this variant as "traditional data profiling" [Nau13]. A conceptual illustration is given in Figure 5.2.

Purpose. The purpose of data-oriented data profiling is to create a data profile, which is subsequently inspected by the user to understand and learn about the data source. This can lead to the discovery of new opportunities or tasks that can be accomplished with the data.

Input. The input can be any kind of dataset, e. g., a database, a flat file, or any other kind of structured or semi-structured dataset. Apart from the data, no other type of input is considered here.

Executor. This variant is executed by an IT professional as characterized in Section 3.5, because it requires a certain degree of expertise regarding metadata and data profiling. The extent of required expertise is dependent on the sophistication of the profiling tool that is used. For example, if a highly sophisticated tool that offers guidance throughout the profiling process is available, then even an inexperienced person can perform this task. The opposite example to this is a case where only low-tech tools are available, like a query interface. Here, the executor would have to write his own queries, which requires a higher degree of knowledge.

Method. The use of a data profiling tool is encouraged to ensure that the resulting data profile is correct and as complete as possible. Additionally, it makes sense to use the same tool for different profiling runs to guarantee that any differences in the results are not due to divergent implementations. This enhances reproducibility and transparency.

If the execution of data profiling is a frequently occurring task within a company, it makes sense to standardize it using a reference process that specifies the individual steps and their order. However, no applicable reference process has emerged so far with any significant amount of adoption, neither in research nor in practice. Thus, the generic profiling model introduced in Figure 4.1 is proposed as a basic template for such an endeavor.

Output. The output of data-oriented data profiling is a data profile that is general-purpose in nature. There is no further specification or requirement placed upon which types of metadata should be included in the profile. Thus, such a data profile should be universal and consist of every metadata type that is applicable to the given input. Applicability considerations here include the data model (e. g., structured versus semi-structured) and the data types of the involved data (e. g., strings versus numbers). It is then up to the user to inspect this profile and look for interesting and useful insights that can be learned.

User. The user is the person that needs to understand and learn about the data source. This can be the same person as the executor, or in larger settings, be a different person that is a domain user instead of an IT professional.

As the user is tasked with inspecting the data profile, he needs to be data literate and know how to interpret metadata. This requires knowledge about what the various metadata types mean, how they are defined, and what can be learned from them about the underlying dataset. Many of these aspects have been laid out in Section 4.2.

Application Areas. Generally speaking, data-oriented data profiling can be applied everywhere where data is involved, because any data source can be profiled. Still, some application areas can be identified in which the use of this technique is most appropriate.

First, data-oriented profiling is appropriate whenever a user is facing a dataset which he is not familiar with. This may for instance be the case when a company hires a new employee, buys a new dataset, or acquires another company and its datasets. Achieving familiarity with a dataset, i. e., having an idea what it contains and how it is structured, is often very desirable, and data-oriented data profiling is the ideal way to do so.

Another application area is data quality assessment, which should be done whenever the general state of the data is in question. This may be the case when the data can potentially be written and updated by a large set of people with varying areas and levels of expertise. Additionally, when the data has not been checked for a long time, it is likely that its quality has decreased. For example, Marsh reports that customer data degenerates by 2% per month [Mar05], which can be attributed to changes in names or addresses that are not reflected in the data. This phenomenon has been described as "changes not captured" [May07, p. 18]. Thus, it makes sense to periodically re-run this profiling variant to verify the overall state of the data.

Example. To give a more concrete example of how data-oriented data profiling can be applied in a practical setting, recall the CloudHost scenario. Here, the overall objective was to design and implement an algorithm that generates sale opportunities based on the company's product portfolio dataset. This dataset contained a list of all CloudHost products, the clients that use them, and in which configuration. As the team that was tasked with the algorithm creation was not familiar with this dataset, more insights about it were required. There was no specific goal yet, as the project stakeholders were still in an exploratory phase, ascertaining what is and is not possible with the data at hand. Thus, it was decided to perform data-oriented data profiling as a first step, using Talend Open Studio for Data Quality (cf. Section 4.4.1) as the data profiling tool. The choice to use this particular tool was based on the fact that it offers an easy-to-use interface, a wide variety of metadata types, and is readily available under an open-source license.

Using this tool, a data profile for the portfolio dataset was created, which consisted of numerous metadata for all included columns. This enabled the team to gain a broad understanding of key characteristics. The main benefit of inspecting the metadata, as opposed to inspecting the raw data itself, is that it can be done much more quickly and unusual patterns or characteristics are easier to spot. For example, it was discovered that the key columns (`product_id` and `client_id`) were of sufficiently high completeness and integrity for the planned analysis. On the other hand, some of the columns had a very high null count (cf. Section 4.2.1), i. e., many of the values were missing. After consulting the domain experts about this, a few of these columns (e. g., rarely used free-form text fields) were discarded for the analysis, as they did not provide any useful information. In another case, it was discovered that the available data about product usage statistics was incomplete due to the fact that it was not mapped properly. With the precise knowledge of which column was affected and to which extent, this issue was easily fixed by re-requesting this data with a proper mapping path. This particular issue prompted the project team to consider multi-source data profiling, which will be described in more detail in Section 5.3.

In conclusion, the data-oriented data profiling helped to kick off the overall project by

Figure 5.3: Goal-oriented data profiling.

providing key insights about the overall state of the data. This allowed issues with the data to be caught early on, instead of later where they would have been much more costly to fix.

5.2 Goal-Oriented Data Profiling

In this second variant of data profiling, the focal point expands from the data to also include a specific goal. This goal is a concrete description of something that needs to be achieved with the data. For example, the goal could be the migration of a dataset from a source system into a target system. This requires checks to make sure that the data fits into the target system, e. g., the number of data instances and fields is less than the supported maximum, the data types of individual fields match the target schema, and all integrity constraints are satisfied. Performing these checks is easy to do with data profiling, because the necessary information is part of the respective data profile. In this example, many metadata types are not required, such as descriptors of the data's distribution. Thus, these metadata do not need to be computed. This omission leads to a partial data profile, which is called *goal-oriented*, because it is trimmed down to satisfy the exact requirements of the goal in question.

A conceptual illustration of this variant is given in Figure 5.3. Next to the data source there is now a second input, which is the aforementioned goal, and the icon depicting the profile contains only partial lines to imply the trimming down.

Purpose. In this variant, the purpose is the accomplishment of the goal. Thus, a thorough understanding of the complete dataset is often not necessary and can be skipped, which leads to savings in time and resources. Instead, the understanding of the dataset is limited to the extent that is necessary with regards to the goal.

Input. There are two inputs: a goal and a data source, which are connected, i. e., the data is necessary for the accomplishment of the goal. The nature of the data source is unchanged, i. e., any kind of dataset is permissible here. The goal on the other hand require a bit more explanation. In the context of requirements engineering, Pohl defines a goal as "an intention with regard to the objectives, properties, or use of the system" [Poh10, p. 53]. The important word here is *intention*, which implies that a goal describes a commitment to taking a specific action in the future. In the previous example case, the goal can be written out as "the data in the source system needs to be migrated into the target system".

No further requirements are placed upon the goal, especially not regarding its format. This means that it can be written in natural language, or given in verbal form, or be expressed in any other suitable form. However, to be operationalized and carried out, a goal needs to

be understandable and sufficiently specific. If that is not the case, e. g., when a client cannot adequately express what exactly he needs, a translation into more suitable terms is necessary. In the example above, the source and target system are not sufficiently specified, which can be remedied by providing details about their names, location, accessibility, and so on. An extensive literature source on how goals should be documented, alongside reference templates and goal modeling frameworks, can be found in [Poh10, p. 103ff].

Executor. Goal-oriented data profiling should be executed by a person that has understood the specified goal. The best candidates for this are usually either the person that has set the goal, or the person that is tasked with its fulfillment. Both of these persons should possess the required knowledge that is needed to derive the necessary metadata types that are used to form the goal-oriented data profile.

Method. Similarly to the data-oriented data profiling variant, the use of a data profiling tool is encouraged. However, instead of simply calculating every possible metadata type, careful consideration should be applied to select those metadata types that are relevant and useful for the goal. Ideally, it should be possible to formulate goals in such a way that the selection of relevant and appropriate metadata types can (at least to some extent) be automated.

Output. The output of goal-oriented data profiling is a goal-oriented data profile. A goal-oriented data profile has the same structure as an ordinary data profile, but it only contains a selected subset of metadata types. The ultimate goal is to provide the information that is necessary to reach the goal, and no other information.

User. The user is the person that inspects the output and derives insight from it. As the output revolves around the fulfillment of the goal, it makes the most sense then that this user is the same person as the one that needs to fulfill said goal. This way it is ensured that the necessary information is supplied to the right person.

Application Areas. As this variant requires a specific goal to be set beforehand, it is less widely applicable as compared to the data-oriented variant. Still, some important areas and scenarios can be highlighted where a goal can clearly be identified.

Data Migration has already been described as part of the introductory example. Another area is data cleaning, in which the objective is to find errors and other data quality issues in a dataset and correct them accordingly [RD00]. These errors range from simple issues (e. g., missing values) to more complex problems (e. g., integrity constraint violations). Goal-oriented data profiling can be applied to specifically look for these issues. This provides the user with an overview of how many issues there are in total and where they are located, which helps in planning measures to address them. As a specific example, when the goal is to deduplicate a dataset, computing the duplicate count as part of the data profile is usually a reasonable first step.

A more technical example of a goal-oriented data profiling application area is *query optimization*, which is performed by a DBMS before queries are executed [Cha98]. A very

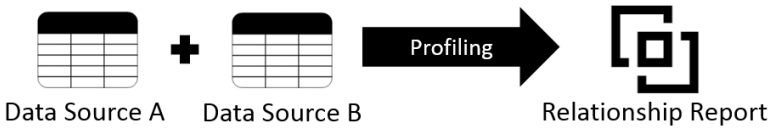

Figure 5.4: Multi-source data profiling.

important metric for query optimization is the cardinality of involved datasets (e. g., the number of rows of a table). For this very reason, most DBMS keep track of all cardinalities in a buffer so that they are readily available when needed and do not have to be computed first. These buffered cardinalities are a prime example of a goal-oriented data profile, because they are kept up-to-date for the explicit goal of optimizing queries.

Example. In the CloudHost scenario, the need for a goal-oriented data profiling emerged after the initial exploratory phase (which contained data-oriented data profiling, cf. Section 5.1). At this stage, the decision was made to implement a customizable recommender system, which should be able to support the sales representatives by generating product recommendations for specific clients. These recommendations should be based on both the existing product portfolio of that client, as well as the portfolio of similar clients. Similarity in this sense should be calculated based on master data about the clients, such as size and revenue.

In order to achieve this goal, it was necessary to ensure that the data at hand was sufficiently able to support the planned recommender system. Thus, another round of data profiling was started, but focused on only those types of metadata that were relevant in the context of that goal. For example, the value distributions in the master client data were of high importance, to ensure that the planned similarity calculation would have the expected results. This was checked by calculating frequency and distribution tables for each considered column. Next, the mappings between clients and products were profiled to assess whether they were of sufficient quality for the recommender. It was found that there were only a negligible amount of foreign key violations (caused mostly by formatting problems and wrong encodings), which were easily cleaned.

The results of all these individual profiling efforts were gathered in a goal-oriented data profile and presented to the stakeholders. This proved to be a valuable basis for a focused discussion about the planned recommender system.

5.3 Multi-Source Data Profiling

So far, only a single data source was considered as input. However, in many cases, two or more data sources need to be compared and inspected for similarities, differences or potential overlaps. This is called multi-source data profiling and its basic concept is shown in Figure 5.4.

Note that multi-source data profiling is not the same as profiling two data sources in sequence, because the focus is not on the individual source's profile. Instead, the parallel profiling of multiple sources at once enables new types of information to be generated, like similarity measures and compatibility metrics.

Purpose. The purpose of multi-source data profiling is to gain insight into the relationship between two or more data sources. This relationship can have different manifestations. For example, it could be the case that the two data sources have the same schema and thus, their data integrates perfectly. If the schemata do not match, but they have a common ID attribute, they could be complementing each other. This is useful when somebody is looking to enhance a given dataset by augmenting it with more attributes from another source. Lastly, it is also possible that the two data sources do not agree on neither schema nor data instances and thus, are incompatible with one another. Finding out the type of relation between the input sources before any resources are invested in integrating them can save a lot of time and money.

Input. As already stated above, the input is a set of two or more data sources, which can be of arbitrary structure. If all sources follow the same data model (e. g., the relational data model), assessing their compatibility through profiling and schema matching is a straight forward task that is well researched. Chapter 5 of the book *Principles of Data Integration* by Doan et al. gives a good overview of this topic [DHI12, p. 121].

The complexity increases when the input data sources have diverging data models, e. g., a relational data source and a document-oriented database. The challenge is that semi-structured databases usually do not enforce a schema on their data, i. e., a schema is only provided *on demand* or *on read*, which complicates schema matching, because it may be different each time it is generated, depending on the use case. There is very little research in this area, a notable example being [BV05].

Executor. This variant requires a high degree of expertise to execute correctly and reliably. There are no easy-to-use tools which perform the necessary steps in an automated fashion at the press of a button, which increases the responsibility of the executor. Thus, only trained staff members should be assigned to this type of task, especially in mission-critical scenarios.

Method. So far, there is no established process for multi-source profiling. However, there are proposed methods for performing data matching. For example, Christen suggests a general data matching process that consists of the steps data pre-processing, indexing, record pair comparison, classification, clerical review, and evaluation [Chr12, p. 24]. Doan et al. propose a matching system architecture that is based on matchers, combiners, constraint enforcers, and match selectors [DHI12, p. 128].

These methods however only consider two data sources at once. Achieving true multi-source profiling with more than two sources at once is still an open research challenge [Chr12, p. 225]. The reason for this is that many basic operations, e. g., comparisons or relational joins, are binary, i. e., they are defined for only two parameters. Extending these operations to more parameters can be achieved conceptually by executing them on pairs and concatenating the results. In practice however, this creates many problems, such as a dependence on the execution order if the operation does not satisfy the associative law. Additionally, it may not be obvious how exactly the results should be concatenated in a meaningful way. Initial work that extends data matching to more than two data sources has been done by Sadinle et al. [SHF11].

Output. The output of multi-source data profiling is a report on the relationship of the input sources, i. e., their differences and similarities. The similarity can be measured on the two levels *schematic fit* and *data fit*. Schematic fit describes how well the schemata of the sources match and can be quantified, e. g., by comparing the number of schema matches with the overall number of schema elements. The data fit on the other hand provides insight into how well the data instances residing in each source match with each other. Calculating the data fit is also known as data matching [Chr12], record linkage [FS69], or entity resolution [Tal11].

In certain cases, the differences might be of more interest than the similarities, and the focus is shifted towards those schema elements and data instances that are *not* a good fit with each other. This can be a very useful exercise when, for example, the two sources are the same database, but from different points in time. Generating a difference report can then reveal what has changed over time.

It must be pointed out that the output of such an effort is best suited to be interpreted manually by a user instead of being used unfiltered in some automated process. Smith et al. write that "in large enterprises involving many information systems, we observe that human planners and decision makers can benefit as primary consumers of the information generated by schema matching, as opposed to these results solely being "piped" into code generation." [SMM+09, p. 6].

User. The similarity report needs to be assessed and reviewed to support decision making. One particular role that can benefit here is the project manager of any data integration project. As multi-source data profiling uncovers how well two data sources fit together, it is particularly well suited for estimating the effort for integration. Abedjan et al. state that "data profiling can [...] assess the integrability or ease of integration of datasets and thus also indicate the necessary integration effort, which is vital to project planning. Integration effort might be expressed in terms of similarity, but also in terms of man-months or in terms of which tools are needed." [AGN15]

Application Areas. The prime example for multi-source data profiling is the preparation of a data integration effort. The purpose of data integration is to combine individual data sources and provide a unified view on all data. This requires that the data sources are comparable to a certain degree, i. e., that they are used to describe similar real-world entities and have an overlap in their data or schema definitions. This overlap is used in a formal process called *schema matching*, during which correspondences between the schema elements (i. e., tables or columns) are documented [RB01]. Data profiling can support schema matching by suggesting likely candidate pairs of corresponding schema elements based on the similarity of their metadata. In other words, if for example two columns have the same data type, similar value statistics, and similar pattern frequencies (i. e., they have similar profiles), then they are likely to be a useful schema match.

Example. The CloudHost project mostly dealt with a single data source, the portfolio dataset. However, in one particular instance, this dataset was augmented with data from

another source about product usage statistics. During the initial data-oriented data profiling (cf. Section 5.1), it was discovered that this data augmentation process was faulty, as many of the expected values were missing. Investigating this issue required an analysis of both data sources: the portfolio dataset and the product usage statistics. As both of these datasets followed the relational data model, multi-source data profiling could be conducted without much effort.

First, it was assessed how the existing augmentation had been set up. It was discovered that a join operation was used, which combines the two datasets on a set of product ID columns. This join was clearly based on the assumption that the respective columns would contain matching data values. However, profiling both columns for their actual data overlap revealed that it was less than 10%, i. e., the amount of data values that appeared in both columns was less than 10%, out of all values in these columns. Later, it came to light that the used product ID column in the product usage statistic dataset was deprecated and no longer updated, causing the observed discrepancy.

Next, a solution to this issue was required, which involved finding a better-suited pair of columns to base the join operation on. The domain experts informed the project team about the correct product ID columns to use in this case. However, before any changes were made, a verification based on the actual data was needed. Thus, another round of profiling was started which assessed the data overlap between the proposed columns. The result was over 90%, which aligned with the stakeholders' expectation and provided confidence that this would indeed be the correct columns to use for a join operation. Consequently, the change was implemented, which significantly improved the usability of the portfolio dataset for the intended analysis.

5.4 Data Profile Validation

The next variant of data profiling is concerned with validating a given data profile and the contained metadata. This can be done whenever metadata is already available for the data source in question, but there are doubts regarding their correctness or accuracy. These doubts can arise for a number of reasons, such as mistrust of the source or expected decay due to age. Olson even goes so far as to assume that all metadata should be doubted. He states: "Data profiling technology starts with the assumption that any available metadata [...] is either wrong or incomplete" [Ols03, p. 122]. This is a very pessimistic approach that mandates a potentially expensive recomputation of existing information, which could be wasteful. However, one could also argue that being overly cautious at this stage is the best way to ensure correct metadata as a basis for future decision making.

No matter what the motivation for validation is, it needs both a data profile and a data source as input, from which a validation report is generated. This process is shown in Figure 5.5.

Note that *data profile validation* should not be confused with *data validation*. As the names imply, the former is about making sure that a given profile is correct, i. e., it contains correct metadata, while the latter is about checking whether the data itself conforms to a set of validation rules, including data types, values ranges, or other constraints. In this sense, data validation is a subprocess to data quality assessment, whereas data profile validation is an

Figure 5.5: Data profile validation.

independent check that can be done before any data profile information is put to use.

Purpose. Data profile validation ensures the correctness of a given data profile by recomputing the metadata from the data source and comparing the results.

Input. The input consists of two parts, a data source and an associated data profile that needs to be validated. The data profile does not need to be complete in any sense and can be a loose collection of metadata. These pre-existing metadata can come from any source. For example, the analysis of a data source's documentation, or an evaluation of interviews with stakeholders and users of the data can result in useful information that should be validated against the actual data.

Executor. Performing this kind of validation requires knowledge about what data profiles are, what they consist of and how they are derived from a dataset. In particular, distinguishing between what is right and wrong in this context requires expertise that only a trained person can be expected to have. Additionally, there are no sophisticated tools on the market that adequately support this task, which further increases the challenge.

Method. The task of validating a data profile as a whole can be subdivided into validating the individual types of metadata that are included. Validating a single piece of metadata is tantamount to recomputing its value and then comparing it to the initial value. For example, if the given metadata states that the row count of a table is equal to 50, this can be validated by executing the query that determines the row count (cf. Section 4.2.1). If the query returns 50, the equality check returns true and this metadata is correct. Otherwise, it is false and an according entry is made in the validation report.

This check of metadata is repeated either for each instance of metadata present in the data profile (complete validation), or just for a subset of metadata instances (partial validation), e. g., only those whose correctness is uncertain or doubtful.

Output. The output of data profile validation is a validation report. It lists each individual metadata validation run that has been executed as well as the respective results. Each entry should include the type of metadata that has been validated, the expected value, the actual value, and the algorithm or query that has been used for the computation. The use of colors is encouraged here to highlight any encountered differences, e. g., in red.

Additionally, the validation report may include an aggregated sum of number of comparisons made and a percentage of how many of them matched. This allows the user to quickly inspect and digest the report.

User. The user is the person that inspects the validation report. In order to do so, he needs to be able to understand and interpret its content, which requires data profiling experience. Additionally, the user is responsible for making decisions and taking actions based on the report. For example, any metadata that is reported as wrong should probably be corrected, which may have implications for other systems that use that data.

Application Areas. Data profile validation should be applied whenever a data profile is being made use of, but its correctness cannot be ascertained. This can occur in many application areas, e. g., when the source of the metadata cannot be trusted, which is often the case when third parties are involved or the data comes from the Web. Another aspect to remember is that metadata is data as well, and thus, is subject to the same circumstances. One such circumstance is that the quality of data diminishes over time, because as data gets older, it is more likely to be outdated. The same applies to metadata, so the older the metadata is, the more skepticism should be applied.

Finally, a data profile can only be expected to be valid for a given input dataset in a specific state. Whenever the dataset changes, e. g., instances are inserted, deleted, or updated, it is reasonable to assume that the metadata changes as well. If any change in the data does not automatically update the metadata, the associated data profile is invalidated immediately.

Example. At the beginning of the CloudHost case, one crucial question was "How many different products does CloudHost have on offer?". It was important to have a precise answer to this question, because it would affect the complexity of the planned recommender algorithm, its scalability requirements and the expected number of resulting recommendations. However, different people had different answers to this question, and they varied wildly. Still, none of them were objectively wrong, because the various departments of CloudHost had different definitions of the term *product*.

For example, the finance department considers the complete Office 365 suite to be one product, because customers usually pay only one bill to use everything it includes. However, the marketing department considers each individual software in that suite as a product in its own right, because they need to advertise each software separately. Further still, the IT department considers each major version of each individual software to be its own product, because they have different requirements regarding the technical environment they need to be run in.

From a data profiling perspective, these reported numbers of products represented metadata that needed to be validated, due to their uncertain correctness. After discussing the various definitions with the project stakeholders, an agreement was reached that a product is anything that has been assigned a unique `product_ID`. Thus, the question of how many different products there are could be answered by calculating the *distinct count* (as laid out in Section 4.2.1) of the product ID column. The result was consequently used to validate the

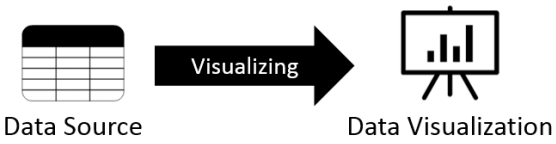

Figure 5.6: Visual data exploration.

previously reported number of products, and reassess some of the initially made complexity estimates, which helped the management of the project.

This brief example shows the validation of a single piece of metadata (here, the number of products). In cases where a proper data profile (i. e., a collection of metadata, cf. Section 4.1) is available, the validation process should be repeated for each individual type of metadata whose correctness is not guaranteed.

5.5 Visual Data Exploration

Data exploration is the process of inspecting a dataset and learning about its contents and characteristics. The techniques that are used for exploration are usually *visual*, which is why it is synonymously referred to as visual data exploration or visual data mining [Kei02].

Data exploration is an important activity for any organization that collects large amounts of data, because data that is merely collected without eventually being put to use serves no business goal while incurring costs due to, e. g., occupying storage space [KA01]. Only when it is known what kind of data is available and how it is structured can it be put to good use. The general process is depicted in Figure 5.6.

The key concept here is the use of visualization techniques to process and present the data. The idiom "a picture is worth a thousand words" can be applied here to stress the fact that humans, unlike computers, can more easily digest information if it is presented in a visual way, e. g., using colors, shapes and other elements [LS87]. This essentially amounts to a translation from a machine-oriented format of bits and bytes into a human-oriented format of visual information.

Another benefit of visual exploration techniques is that they are designed with interactivity in mind. This means that the resulting visualizations are not static, but can dynamically be adapted by the user, enabling navigation through the data. This is especially useful when there are too many data instances to be viewed at once, as visualizations aggregate and summarize the data in a compact manner.

Data exploration is linked to data profiling, and consequently considered here as a data profiling variant, because it is based on the same types of metadata. In fact, many exploration techniques require that a specific type of metadata is calculated first, and then piped into a visualization technique. From this perspective, data exploration extends data profiling by adding this final visualization step to the process. It could be argued that it should thus be called *visual data profiling* instead. However, the term *visual data exploration* is already established in the literature, which is why it is kept as a name for this variant.

Purpose. The purpose of visual data exploration is to enable a human user to use his cognitive visual ability to make sense of a dataset.

Input. The input consists of a single data source with any kind of structure. If the data source is small, i. e., it has a low number of data instances and fields, it is usually sufficient to look at it directly with an appropriate tool, e. g., spreadsheet software for tabular data. This may have the benefit that it is unfiltered and enables immediate insight into the available data. This approach quickly fails when the size of the data increases even slightly to just a few hundred components, because a human reader cannot reliably process that much information at once. Thus, visual data exploration is recommended on input data sources that are moderate to very large in size.

The data type of the input data plays a crucial role for its visualization, because the set of applicable visualization techniques are dependent on it. The term *data type* is used slightly different in the context of visualization as compared to the technical perspective introduced in Section 4.2.1, where basic types such as characters and numbers were considered. According to Shneiderman, the following seven data types can be distinguished for visualization purposes [Shn96]:

1. One-dimensional: linear data type, organized in a sequential manner. E. g., documents, lists of data.
2. Two-dimensional: planar or map-type data. Used for geographical and location-based systems.
3. Three-dimensional: describes real-world objects and their dimensions, e. g., buildings. Used in computer graphics and computer-assisted design.
4. Multi-dimensional: arbitrary number of dimensions. Relational tables fall in this category.
5. Temporal: data that has explicit timestamps for start and end. Used in logging, project management, or medical documentations.
6. Tree: data that is structured such that every item has a link to one parent item, except the root. Useful to describe hierarchies.
7. Network: graph-based data, where every item can be linked to any other item. Used to show social networks.

Each of these data types can be visualized with appropriate techniques. For example, if the data input resembles a social network, then graph visualization methods can be applied [HMM00].

Executor. Exploration is always performed using a suitable tool, so the executor of the exploration is automatically the user of that tool. As the usefulness of data exploration tools has been recognized early on [dOL03], the market for these tools is very mature and the sophistication levels accordingly high. This leads to a wide availability of easy-to-use tools that are accessible to anyone that knows how to operate a computer. Consequentially, the entry barriers into this discipline are relatively low.

Method. Visual data exploration is an interactive and iterative process that is dependent on support through an appropriate tool. Such tools allow users to navigate through the dataset and explore it, which is complemented by suitable types of interaction. These interaction types have been categorized together with their respective user intent. The following list shows one such categorization [YaKS07].

- **Select** Mark something as interesting
- **Explore** Show me something else
- **Reconfigure** Show me a different arrangement
- **Encode** Show me a different representation
- **Abstract / Elaborate** Show me more or less detail
- **Filter** Show me something conditionally
- **Connect** Show me related items

For a more high-level perspective on how visual data exploration should be structured, the visual information-seeking mantra proposed by Shneiderman offers useful guidance. This mantra states "Overview first, zoom and filter, then details-on-demand" [Shn96, p. 337], which is an intuitive approach for visualizing data in a top-down fashion.

A frequent problem in data visualization is that it is a time-consuming and costly task to construct a suitable visualization that reveals interesting insights about the data that can actually be understood by the reader. Thus, a method to automatically generate visualizations that are useful for data profiling can be highly beneficial. One such method is called *Profiler* and is described by Kandel et al. [KPP+12]. It offers an extensible system architecture, an automatic view suggestion, and scalable summary visualizations.

Output. Seeing how visual data exploration is considered an interactive process, there is no immediate need for a tangible output besides what is put on the monitor, because the user is learning what he needs to know while exploring, which already achieves the purpose. Still, it makes sense in many cases to consider exporting static images of the current visualization. These visualizations can be used to reproduce the exploration process and educate a broader audience about the findings, e. g., in a presentation.

User. The user is either the same person as the executor in cases where he is exploring the data on his own. The group of users expands accordingly in cases where the exploration results are shown to other people.

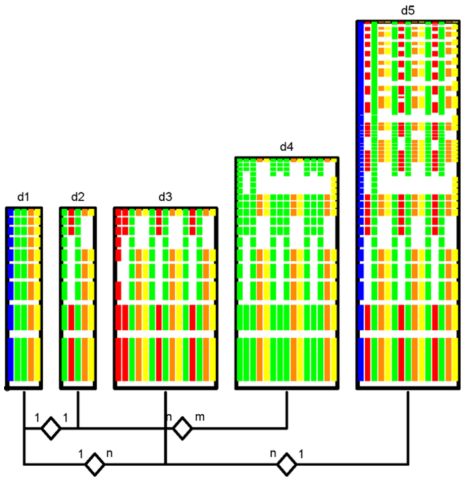

Figure 5.7: CityPlot example. Source: [DV12].

Application Areas. The usage of visual data exploration techniques has been proposed for a diverse range of application areas. For example, von Zernichow and Roman present their tool *Grafterizer* in [vZR17], which utilizes visualization techniques to facilitate data cleaning and transformation.

Another popular tool is *Gephi* [37], which focuses on graph and network data and offers a powerful engine for visualization, arrangement and manipulation. Gephi is advertised as a tool for exploratory data analysis, which is a term that has its roots in statistics [Tuk77], but means essentially the same as visual data exploration. Gephi enables application areas such as social network analysis or the visualization of complex scientific data from domains such as biology [OHB12, VKH15] or chemistry [CLD+12].

Lastly, visual data exploration can also be applied to inspect relational data. One noteworthy tool as an example for this is *CityPlot*, which generates a skyline-like representation of the input data [DV12]. This allows a quick visual assessment of table cardinalities and their relation to each other. An example output of CityPlot is shown in Figure 5.7. The colors in this example denote the data type of the respective data cell, with numeric data being blue, categorical data green, date/time yellow, and others in red. White spots show missing data, i. e., null values.

These are just some examples of visualization application areas and corresponding tools. There exist many more and they cannot all be listed here. The interested reader is referred to surveys on the matter, such as [LCWL14] or [BK01].

Example. In the CloudHost project, there was one particular instance where a visual exploration of the data had been key to making progress. Remember the portfolio dataset, which used a relational data model and had been made available as a spreadsheet file. Some of the preliminary analyses on this dataset that have been conducted for test purposes yielded unexpected findings, i. e., some of the aggregation results were outside the range of expected values. After double-checking the analysis implementation and finding no issues there, suspicions were raised that something was wrong with the data itself.

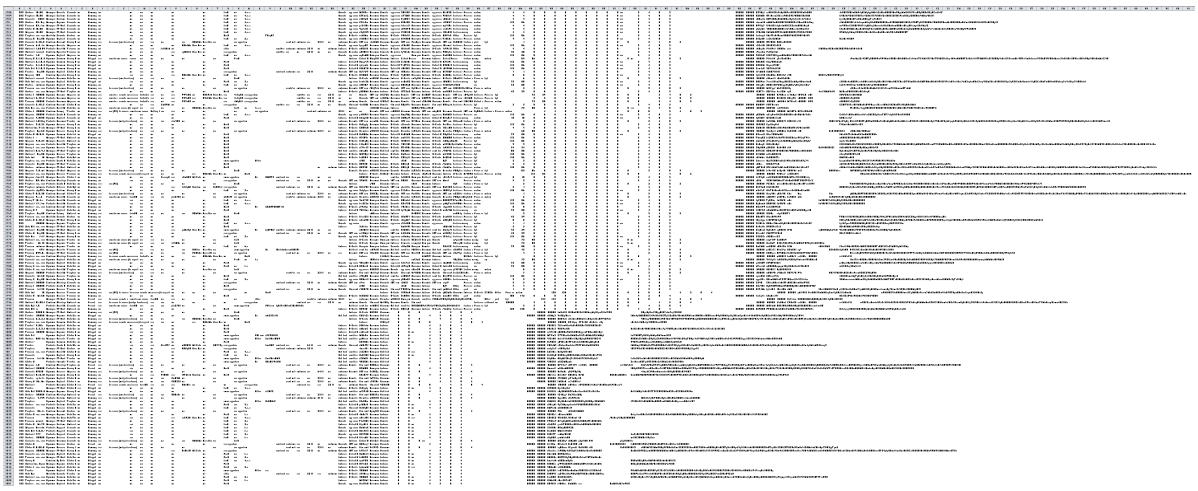

Figure 5.8: Spreadsheet view of the CloudHost portfolio dataset.

The natural reaction at this point was to open the file in a spreadsheet software and simply *look* at the data. The default setting of most spreadsheet software is to display data in a readable size, which means that only a handful of data values fit on one screen and are visible to the user. The remaining data can only be seen by scrolling horizontally (to see more columns) or vertically (to see more rows). Looking for anomalies or irregularities in the data while alternating between these scrolling mechanisms is similar to searching the proverbial needle in a haystack, i. e., infeasible.

As pointed out before, the first decree of the visual information-seeking mantra is "Overview first" [Shn96, p. 337]. In an effort to gain an overview of the data, the zoom functionality of the spreadsheet software was used to reduce the display size of the data significantly, so that at least all of the columns could fit on the screen. Although this caused the actual data values to be no longer legible, it provided a much better overview of the structure and general form of the data. The horizontal scrollbar was now eliminated, which significantly reduced the complexity of going through all rows and looking for irregularities. Near the end of the file, the data looked like shown in Figure 5.8.

In the lower half of that view, the structure of the data changes and the values seem to be shifted to the left. Investigating this anomaly revealed that all rows below that point have different columns compared to the rows above. 17 columns are not present in these rows, which had been loaded through a different (outdated) process and appended to the file. When encountering such a format mismatch, there are generally two options: discard the offending data, or correct the issue at the source. In the CloudHost case, a correction was possible, which involved modifying the loading process with an updated column list and re-executing it. This resulted in a cleaner dataset, which no longer exhibited the unexpected results that had initially prompted this investigation.

Even though this example employed a very basic form of data visualization, it could still be demonstrated why visually inspecting a dataset can be very valuable: Structural anomalies or irregularities in a dataset tend to stand out and are easily spotted by the human eye. Identifying and dealing with such issues is usually very beneficial for any data-driven project,

Figure 5.9: Data discovery.

especially early on before they cause any complications in the downstream processing of the data. Furthermore, the use of more sophisticated data visualization tools and techniques can reveal more obscure issues in more complex datasets.

5.6 Data Discovery

The last data profiling variant described in this chapter is data discovery. Data discovery is done to search through a space of available data to find a dataset that satisfies a given set of requirements and criteria. This search space in which the discovery takes place can either be internal, e. g., a data repository or a data lake, or external, e. g., the Internet. The search criteria that describe the desired dataset are descriptions of its characteristics and thus can be considered metadata themselves. This enables the treatment of a set of search criteria as a data profile. This data profile then assumes the same role as the search parameters in a traditional search process. This constellation is shown in Figure 5.9.

Data discovery can be seen as some kind of *reverse data profiling*, because it reverses the input and output objects. However, instead of *generating* the output from the input, it is rather *discovered and found.*

Note that there is a degree of linguistic similarity between the words *exploration* and *discovery* as both describe acts of investigating and searching the unknown. However, there is a notable distinction between *data exploration* and *data discovery*, as the former is about investigating a dataset that is already available, while the latter describes a method to find one dataset among many that satisfies a set of criteria.

Purpose. The purpose of data discovery is to find one or more dataset(s) within a search space. This can be useful if, for example, new data is needed to fulfill a specific purpose like enabling an analysis or augmenting another dataset.

Input. The input for data discovery are the search criteria given as a data profile. There are two ways in which such data profiles can be generated. First, they can be defined manually by specifying the metadata and their values. These hand-crafted data profiles allow a maximum of flexibility, because there are no restrictions regarding their contents. The second way is to derive a data profile from the profiling process of another dataset. This enables a discovery of datasets that share the same metadata and can thus be assumed to be similar in some way.

Another implicit input to the discovery process is the definition of the search space. This tells the executing system *where* it should look. For the discovery to be successful, it is

important that the datasets available in the search space are sufficiently described through appropriate metadata.

Executor. The decision who should be responsible for executing data discovery depends on the application for which the data is needed. For example, a newspaper article about the most recent weather phenomenon might be improved by a comparison with historical weather data. Looking for this historical weather dataset is a discovery process with a low complexity, because it can probably be easily accomplished by searching for its title. Thus, the executor can be anybody with basic search skills.

Another example with a higher complexity would be if a company is looking to augment its data warehouse with more detailed customer data. This data needs to be compatible with existing customer records and give insight into a specific set of product preferences. Also, the dataset will be used in a core system of the company, hence its quality and trustworthiness are of high importance. Discovering such a dataset involves a precise definition of a corresponding data profile, which is an elaborate task that should be performed by trained IT professionals.

Method. Applicable methods are largely dictated by the search space. If, for example, one wants to search through the whole Internet, the only reasonable possibility is to use a general-purpose search engine like Google or Bing. These search engines do not allow to enter specific metadata to filter and refine the search, so it is very broad in nature.

Another kind of search space are data repositories, which are manually curated lists of datasets that serve a specific purpose. For example, many governments have an open data program in which they offer data about topics such as public spending or education, such as Germany [10] and the United States [40]. These data repositories can be freely browsed through a Web interface, which allow a specific set of search parameters to be set, such as data format, topic, tags, or publisher.

Lastly, data discovery can be heavily facilitated through the usage of a specialized data marketplace as the search space. A data marketplace is a platform that allows anybody to offer their datasets, either for free or a specific price [SSV12]. These data marketplaces have a vast amount of datasets to offer due to their open and collaborative nature and are thus an ideal search space for data discovery. However, many such marketplaces do not offer very comprehensive search mechanisms and instead rely on key-word search and categorization of datasets. There is room for improvements in this area, because a more sophisticated search interface would enable new business cases such as automated data discovery or applying recommendation engines to datasets.

Output. The output of the discovery process is the desired dataset, or at least a location identifier where it can be found. Alternatively, the output can also consist of a subset of a found dataset that is the result of applying a specific filter. This output is expected to fulfill the criteria specified in the input data profile. If the trustworthiness of the data source is doubted, a subsequent data profile validation process can be initiated to make sure that profile and dataset match.

Table 5.1: Data profile for the data discovery of the portfolio dataset.

Metadata	Value
Column Names	`client_id, product_id`
Row Count	> 20'0000
Distinct Count of `client_id`	> 200
Distinct Count of `product_id`	> 1000

User. Usually, the user that needs the data is the same person as the one executing the discovery process. However, there can also be scenarios in which somebody is tasked with the discovery of a dataset for somebody else. In this case, the user can be anybody that uses that dataset.

Application Areas. The biggest application area for data discovery is data acquisition, i. e., when data needs to be sourced from a third party, either via payment or for free. In these cases, a precise purpose can be specified, translated into a data profile, and used as input to the discovery process. As pointed out above, this line of thinking can be applied to open data repositories or data marketplaces.

Another area is when a company is collecting vast amounts of data from various sources and stores them without proper documentation, e. g., in a data lake. Deriving value from such an unorganized collection of data tends to be very difficult due to the sheer volume and variety involved. This can be facilitated by automatically extracting metadata about all included datasets and compiling them in comprehensive data profiles. Then, all datasets can be effectively and efficiently be searched, allowing a successful data discovery, and ultimately enabling value to be gained out of it.

Example. The previous cases from the CloudHost scenario assumed a dataset to be already available, most notably the portfolio dataset. Before these cases had been started, however, some of the involved datasets needed to be found, i. e., *discovered*, first. This section describes how the portfolio dataset had been searched for and ultimately discovered.

Based on the idea to create an algorithm for automated client-specific product recommendations, a dataset was needed that contained information about both clients and the products they are using. Thus, the first requirement imposed on the dataset was that it should contain columns to uniquely and unambiguously identify clients and products. Such identifying columns were expected to be named `client_ID` and `product_ID`, respectively. Next, the dataset must have a sufficient amount of data to allow a meaningful analysis. After some discussion, it was decided that a minimum of 20'000 rows would be necessary to achieve that. Lastly, the data should cover a large portion of CloudHost's clients and products, to ensure that the envisioned results would cover a wide amount of the business. This notion was concretized to at least 200 different clients and 1000 different products.

With the requirements specified, they could be translated into a data profile, as shown in Table 5.1. This data profile constituted the starting point for the following data discovery.

With the data profile specified, a search space for the discovery needed to be determined. It was decided that the CloudHost-internal data lake is the only reasonable candidate for this. As the needed data is by definition internal to CloudHost's operations, it did not make much sense to include any external sources in the search space. The data lake was a loose collection of many different kinds of datasets across the whole enterprise. It had a basic search interface, which allowed a user to look up specific schema elements of the relational datasets it contained, like table and column names. With this interface, a search was conducted for datasets that contain both required columns, `client_id` and `product_id`.

The search found 13 datasets. For each dataset, the row count, as well as the distinct count of both columns was calculated using a data profiling tool. Only two datasets fulfilled the specified criteria. Both of these candidates were investigated manually to decide which one would be best suited for the planned goals. In the end, the portfolio dataset has been selected, which consequently acted as the input for other examples of profiling techniques throughout this thesis. With the desired dataset found, the data discovery ended.

This example demonstrates how data discovery can be executed in a practical setting, from requirement gathering, over search space evaluation, to result selection. When all these steps are properly documented, this approach can lead to transparent and reproducible results, and contribute to the overall success of a data-driven project.

5.7 Summary

To conclude this section, all the presented data profiling variants are condensed into one super process, shown in Figure 5.10. It shows each variant as an activity with its corresponding inputs and outputs.

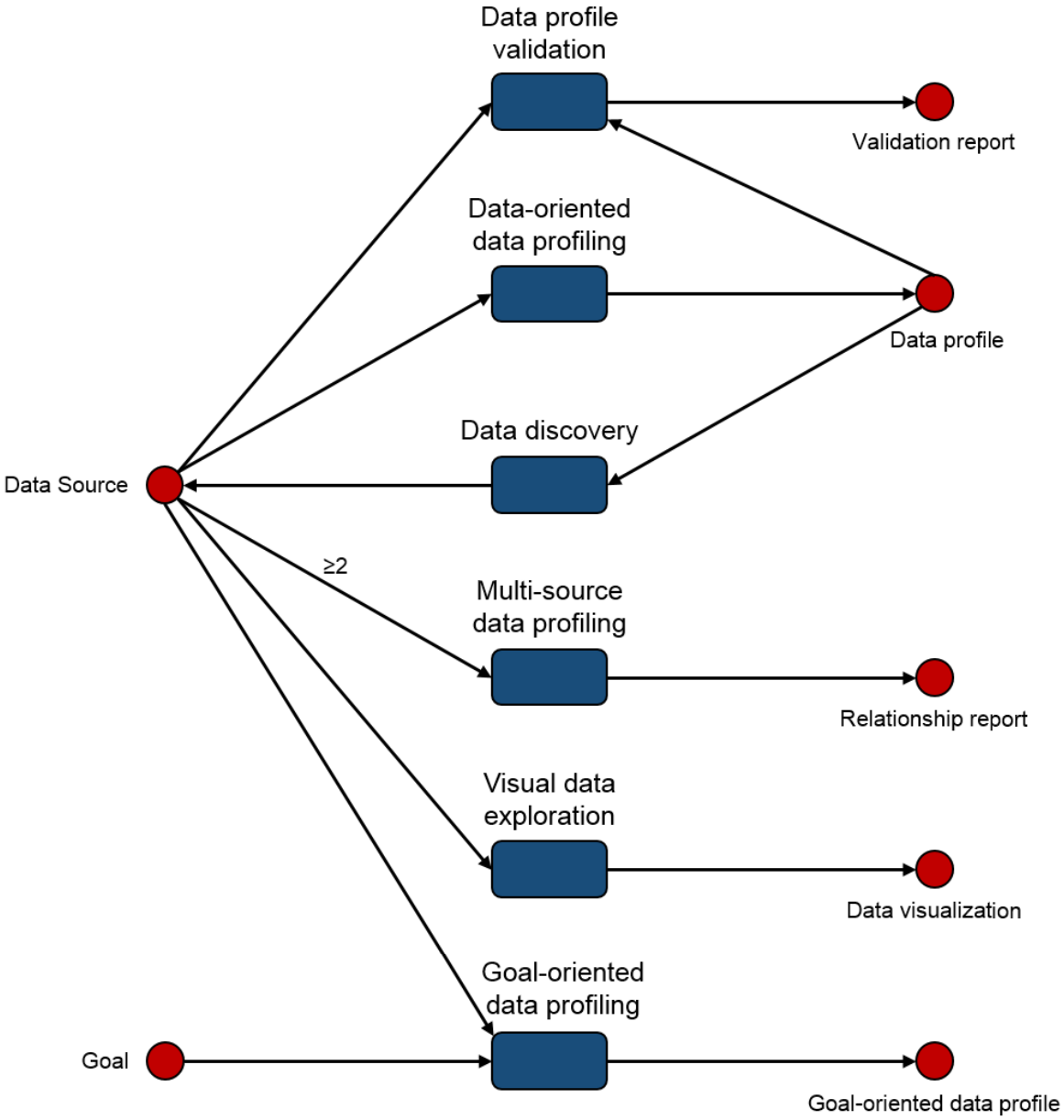

Figure 5.10: Data profiling super process.

Part II

Beyond Data Profiling

6 Broadening the Scope of Metadata

So far, one of the core assumptions was that metadata can be directly derived from an input dataset. However, upon closer examination of the issue, it becomes evident that this is not always the case. There exist types of metadata that relate to facts that are not part of the data itself. For example, the origin of a dataset and the transformations it has undergone up until this point, i. e., the *data provenance* [BKWC01], are information about that data which cannot be learned from the data itself. Instead, the provenance information may be hidden away in log files, or be only implicitly known by the people that have worked on the data before. Still, it is a type of metadata, because it describes a property of the dataset and thus, fits the definition of "data about data".

The fact that there are two distinct sources from which metadata can be derived, i. e., the data itself on one hand and other external sources on the other hand, suggests that a clear distinction between these classes of metadata types should be established. This is done in Section 6.1. To demonstrate the usefulness of this concept, an example case for this chapter is introduced in Section 6.2, which will be used to explain these individual types of metadata in Section 6.3. Afterwards, the two questions *where* such metadata can be found and *how* it can then be extracted are answered in Section 6.4. Lastly, the process of extracting extrinsic metadata is combined with the the process of extracting intrinsic metadata in Section 6.5 in order to construct a combined process.

6.1 Distinguishing Intrinsic and Extrinsic Metadata

The metadata types that have been discussed in-depth in Chapter 4 can all be derived directly from the data. In other words, the information is already present within the data and only needs to be gathered and extracted by an algorithm. This is the first class of metadata types, which shall be called *intrinsic metadata* henceforth. The term *intrinsic* is used to highlight the fact that these types of metadata are based on properties that are inherent to the data in question and require no external source. In other words, this means that there is a direct functional relationship between the input data and its intrinsic metadata, which can be expressed in terms of an algorithm. In conclusion, the following statement is established:

Intrinsic Metadata is metadata that can be derived solely from the data itself.

Metadata types that do not conform to the above statement relate to properties outside of the data. Consequently, it is proposed to use the antonym of intrinsic, *extrinsic*, to label these types of metadata:

Extrinsic Metadata is metadata that cannot be derived solely from the data itself.

This complementary definition of classes leads to a binary distinction, i. e., any metadata type is considered to be either intrinsic or extrinsic and there exists no third class. All metadata types that have been discussed in Chapter 4 are intrinsic. In particular, the metadata extraction process shown in Figure 4.4 can now be considered to be an *intrinsic* metadata extraction process.

As a side note, it should be commented that this binary classification might be too strict, depending on the context and use case. It could be possible to conceive a more general concept where these two types are the end points of a continuous spectrum, and in between would be types of metadata that are "a little" extrinsic or "a bit" intrinsic. For example, the age of a given piece of data could be considered intrinsic in cases where it is explicitly stored as part of the data, and extrinsic otherwise. For the scope of this work however, it is more useful to keep the binary restriction, because it allows a more focussed discussion on the individual types of metadata and from which source they can be retrieved.

Including extrinsic metadata as part of a data profile can be highly advantageous, because it provides a more thorough and holistic picture of the dataset in question. A real-world example of a successful company that explicitly considers extrinsic information in their operations can be found at Amazon. While they perform extensive analyses on the data they have available, i. e., intrinsic metadata, the CEO, Jeff Bezos, has made it a point to explicitly listen and react to the comments and anecdotes of their customers. In a recent interview, he said "the thing I have noticed is when the anecdotes and the data disagree, the anecdotes are usually right" [2]. This demonstrates that not everything can be explained by numbers and algorithms, and it can be beneficial to look beyond the data and search for extrinsic metadata.

6.2 Revisiting the CloudHost Case

To better illustrate the individual types of extrinsic metadata and how they are useful with regards to a specific dataset, the familiar CloudHost example case will be revisited (cf. Section 1.3). Remember that CloudHost is a company that rents out server capacities for customers to run software on. All sales made by CloudHost are recorded in a database table, which consists of the following seven columns:

id Unique identifier for each sale
customer Name of the customer that bought an item
item Name of the item that has been bought
price Amount of money that has been paid
discounted Indicator of whether the sale price has been discounted
registration_fee First-time customers get charged with a one-time registration fee upon their first purchase from CloudHost. This fee is sometimes waived, e. g., during special promotions.
date Date of the sale

Across the years of CloudHost's existence, numerous sales have been recorded. A small excerpt of this data is given in Table 6.1. Note that although this example is based on a

Table 6.1: Excerpt of the CloudHost sales data.

id	customer	item	price	discounted	registration_fee	date
				⋮		
47	Customer01	A	$700	No	$50	2018-05-05
48	Customer01	B	$650		⊥	2018-05-06
49	Customer02	A	$500	Yes	$0	2018-05-08
				⋮		

relational data model, most of the following concepts are easily transferable to other data models, e. g., semi-structured models.

The goal in this example case is to integrate the sales data into the CloudHost data warehouse. There are two users in this scenario, namely the *IT professional* and the *domain expert* (cf. also Section 3.5). The IT professional is responsible for the integration process and has to implement and execute it. He has already performed goal-oriented data profiling, in a similar fashion as described in Section 5.2. Thus, he already has all the necessary intrinsic metadata he needs for his task. What he is lacking though is information about the surrounding context of the data to make better sense of it. In other words, he needs extrinsic metadata, which he does not have immediate access to. This is where the domain expert comes into play: he is familiar with the dataset and how it is used, and thus has implicit knowledge about it. It is further assumed that the domain expert is willing to cooperate and answer any questions the IT professional has. A visualization of this scenario is given in Figure 6.1, which is intentionally kept general to make it applicable to other, similar use cases.

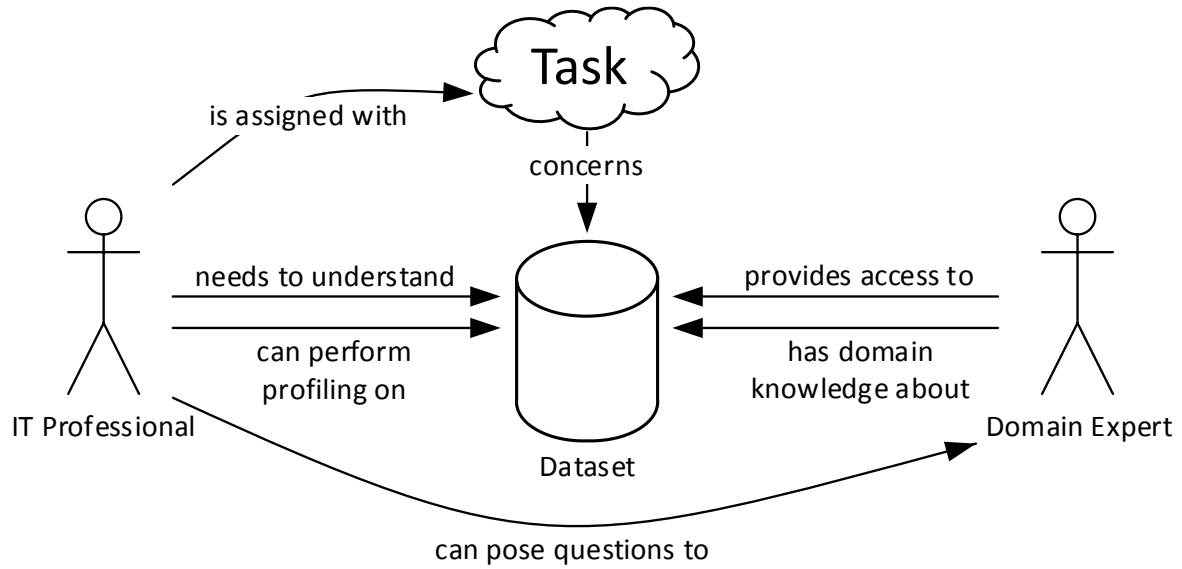

Figure 6.1: Use case diagram for the sales example.

6.3 Types of Extrinsic Metadata

There is a number of different types of extrinsic metadata that can be used to gain a deeper understanding of an associated dataset. This section aims to give a comprehensive, but not exhaustive overview of the most important types according to their anticipated usefulness in a data-driven project. Unlike the list of intrinsic metadata, there are no algorithmic definitions of these types as they are not applicable here. Instead, a higher-level conceptual overview of what each metadata type entails is given, as well as its application in the context of the sales example case.

Data Provenance *Data provenance*, also referred to as *data lineage* [WS97] or *data pedigree* [Rom99], has been characterized as "the description of the origins of a piece of data and the process by which it arrived in a database" [BKWC01, p. 1]. This definition is a combination of two distinct concepts: data origin and data transformation history.

The data origin indicates where the data comes from, i. e., what were the circumstances that led to its inception. There are two broad classes of data origin which can be distinguished:

- *Machine-generated* data is "data that is generated as a result of a decision of an independent computational agent or a measurement of an event that is not caused by a human action" [1]. Examples include computer and network logs, sensor readings and satellite data.
- *Human-generated* data is a recording of the "direct result of human choices" [20]. Examples include e-mails, word files, but also records about purchases, payments, inquiries, comments, evaluations, and notes.

The data origin type allows to draw conclusions about various aspects of the data, such as the expected velocity: Machine-generated data can be produced at enormous speeds, which continually rise along with the computational power of the machines. Human-generated data on the other hand grows much slower, as humans have a cognitive limit on how fast they can make reasonable decisions. The expected growth rate of a dataset is important, e. g., for the adequate sizing of the systems that shall process it.

The second part of data provenance, the data transformation history, is an (ideally) complete list of transformations that have been applied to the data since its inception. These transformation range from simple computations like copying or aggregating to complex data management tasks like data cleansing, data integration, or data curation. One way to represent the transformation history of data is a directed acyclic graph that links the data to the performed activities and the respective agents [CCM17].

The provenance of data is crucial in applications that require transparency and the ability to precisely keep track of a dataset. Examples of such applications include scientific data management for experimental results, the creation of intellectual property, or regulations that involve audit trails. Furthermore, data provenance is universally useful in scenarios where "data debugging" is required, i. e., the possibility to go back and trace any specific point of data. Overall, it can be used to facilitate the assessment of the quality, reliability, or trustworthiness of a dataset [CCM17].

Table 6.2: List of permissible privileges in MySQL. Source: [22].

ALTER	EVENT	RELOAD
ALTER_ROUTINE	EXECUTE	REPLICATION_CLIENT
CREATE	FILE	REPLICATION_SLAVE
CREATE_ROUTINE	GRANT_OPTION	SELECT
CREATE_TABLESPACE	INDEX	SHOW_DATABASES
CREATE_TEMPORARY_TABLES	INSERT	SHOW_VIEW
CREATE_USER	LOCK_TABLES	SHUTDOWN
CREATE_VIEW	PROCESS	SUPER
DELETE	PROXY	TRIGGER
DROP	REFERENCES	UPDATE

Applying these concepts to the CloudHost case reveals the following: First, the data recorded is about sales, which are human choices, so it is human-generated data. This leads the IT professional to conclude that the existing ETL system will likely suffice for the integration task and no expensive big data technology is required. Second, the domain expert tells the IT professional that the dataset is a straight export from the CRM system, with no further transformation or curation being applied. This indicates the need to perform thorough data quality checks before the data can be loaded into the data warehouse, which are implemented as part of the integration process.

Data Privileges A privilege is the right to perform a specific operation on a (subset of the) dataset. Privileges are used to precisely govern which user (or user group) may access and manipulate which part of the data. The range of privileges depends on the functionality offered by the underlying database management system. The most basic operations that any DBMS offers are create, read, update and delete, often abbreviated as CRUD. Popular DBMSs offer many more operations and, consequently, a much more fine-grained privilege system. For example, MySQL offers a total of 30 different types of privileges, as shown in Table 6.2 [22].

Privileges can be either granted directly to individual user accounts, or to a *role*. A role in this context is a middle layer between users and privileges that can be used to facilitate privilege management. Users are associated with roles and automatically inherit all privileges that have been granted to that role in addition to user-individual privileges. Roles are supported by most modern DBMSs.

Related to data privileges is the concept of *data ownership*. Simply put, the data owner has the right to grant and revoke privileges to others [VBM95]. This is also referred to as having *administrative privileges*.

When a user is working with a dataset it is usually very helpful to know which privileges he has and who can change them, i. e., who the data owner is, because it tells the user exactly what he can and cannot do. If, for example, a user is tasked with performing data cleaning, but does not have the privileges to update the data, he cannot accomplish this task. Thus, he has to turn to the data owner and ask for the appropriate privileges to be granted to him.

In the CloudHost example, the dataset resides in a strictly governed production database. The IT professional may only read the data, but not manipulate it in any way. However, when copying the dataset, e. g., in an ETL process to move it into the corporate data warehouse, it is possible to transform, enrich, or augment the data.

Data Location Every dataset needs to be stored somewhere. This *somewhere* is the data location and can be specified in a number of ways. On the Internet, the usage of URLs (*uniform resource locator*) has been established as the de-facto standard for specifying the location of a resource. In this context, the term *resource* refers to a Web resource which can be anything with an identity, ranging from files over services to datasets. In offline scenarios, the location of a dataset may also be specified using an ordinary path in a file system, or by pointing to a specific data storage device, such as a USB flash drive.

In many cases, the data location is conveyed implicitly without a need for further clarification. However, there are also cases in which a specific statement of this metadata type is useful. For example, when a collaborative project is set up and one party agrees to give data to another party, it needs to be made clear in the project documentation when and where this data can be accessed. Doing this upfront saves any unnecessary effort put into searching for the data later on.

There may be cases in which the data location is known, but it cannot be accessed. A classification of reasons why data may be inaccessible is given by Pyle and consists of the following categories [Pyl99, p. 118]: legal issues, departmental restrictions, political reasons, wrong data format, lacking connectivity, architectural reasons, and timing issues. If the data location is not accessible, it is advised to investigate the reason in order to find a solution.

So far, the data location has been considered from an access-oriented point of view. However, it may also make sense to think of the *physical* location of the data, i. e., where the data center that stores the data is. For legal reasons, it may be a strict requirement that data is stored and processed in a specific country. This is an issue that is especially relevant when outsourcing data storage to a cloud computing provider.

The domain expert in the CloudHost example informs the IT professional about the database that contains the dataset and under which IP address and port number he can access it. The IT professional uses this information to configure his tools correctly and ensure successful data processing.

Data Completeness Data completeness is a measure that describes how many of the expected entities and attributes are present in a given dataset. Pipino et al. refer to this concept as "schema completeness" [PLW02]. It is a two-dimensional concept that can further be broken down into entity completeness and attribute completeness. To compute the entity completeness, the number of observed entities (or rows, data instances, etc.) in a dataset is divided by the number of expected entities. This in turn requires knowledge about the number of expected entities, i. e., how many entities the dataset *should* have. For example, if a dataset describing the states of Germany only contains 14 rows (cf. *row count* in Section 4.2.1), while it is known that there are 16 states, the entity completeness can be said to be $14/16 = 0.875$. Similarly, the attribute completeness measures how many of the expected attributes (or

columns, fields, etc.) are available in the dataset.

Both dimensions of data completeness require knowledge of what the respective expected counts are. This knowledge usually comes from an external source (like the number of states in Germany, which can, e. g., be looked up from Wikipedia). Thus, data completeness cannot be derived from the data itself and fulfills the definition of extrinsic metadata.

The idea of completeness is related to the concepts of closed-world assumption (CWA) and open-world assumption (OWA). Under CWA, all real-world facts are contained in the data and non-existing facts are assumed to be false [Rei78]. The CWA can be made if data completeness is fulfilled. As an example, consider an airline database that lists all flight connections for a specific airline. If the source can be trusted, it is reasonable to assume that all flight connections are indeed contained in the database, and if a specific connection cannot be found, it does not exist.

The OWA on the other hand states that missing facts in a database can be either true or false and cannot be verified. For example, a telephone book should usually be treated with the OWA, because it cannot be known whether persons not listed do not have a telephone number, or whether they simply opted out of having their number published. Thus, it is unreasonable to assume that a telephone book is data complete.

Note that the concept of data completeness is, just like data quality, dependent on the use case and context. Wang describes completeness as "the extent to which data are of sufficient breadth, depth, and scope for the task at hand" [WS96]. If a user is looking for a flight by a specific airline, he may have a complete dataset at hand, but if he is looking for *all* flights, the same dataset would be considered incomplete. Thus, it is important to always define completeness with respect to the context in which it is assessed.

Knowledge about data completeness is useful in a variety of ways. If it is known that a dataset is complete, then no further sources are needed to complement it. On the other hand, if it is incomplete, then it might be worthwhile to find out in which regard it is lacking, i. e., which entities or attributes are missing. Furthermore, the state of completeness influences the analyses that can be performed on the data. For example, the maximum value of an attribute becomes meaningless if a significant portion of the values are missing, whereas the average value is more robust in this regard.

In cases where it is known that a dataset is incomplete it makes sense to ask for the reason *why* that is the case. Depending on the individual reason, there may or may not be measures that can be taken. For example, if the survey responses of an interviewee are not stored properly and get lost, the data has never been recorded. In such a case, the data can be recovered by asking the same person again. This is different from cases where reproducing the data creation process is not possible, e. g., when the time is a crucial component of it, like weather sensors that record the temperature and humidity. Such data cannot easily be reproduced, because the specific circumstances have irrevocably passed.

Another reason for missing data is that, after it has been recorded, it is deleted intentionally. Deletion can occur along any level of the data, be it tables, entities, attributes or any other group of values. If data is deleted, it might be worthwhile to investigate why that is the case. For example, there could be data privacy laws that need to be complied with, which leads to the omission of specific values so that individual persons cannot be identified from the data. This is a frequent occurrence when medical patient data should be published without

infringing on the individuals right to privacy. Solutions to this challenge are studied under the name *differential privacy*. A comprehensive overview of this topic is given by Dwork [Dwo08]. If data has been deleted intentionally, the user working on that data can usually assume that there is nothing he can do and cease any efforts to reach a higher degree of completeness. Other reasons for intentional data deletion include negligence, i. e., somebody thinks parts of the data are not important, or size restrictions, e. g., data needs to fit into a physical drive or the attachment of an email and is trimmed down to make it fit.

If the data has been recorded properly and was not deleted intentionally, there may be a third reason for incomplete data: it is located somewhere else. This can happen on a technical level, e. g., data is stored in a distributed database and some nodes go offline, making parts of the data inaccessible. Or it can happen on an organizational level, e. g., somebody misplaced a spreadsheet file in the wrong directory, or gave it the wrong name. These issues are related to the concept of data location (cf. previous section) and can usually be fixed with reasonable effort.

Back in the CloudHost sales example, the IT professional learns that the provided dataset only contains sales made in Europe. Thus, it is a complete recording of all sales that have occurred in that continent, but it does not contain any sales from other continents. This fact should be recorded in the documentation of the dataset. Any reports or analyses based on that data should be made with that fact in mind. To reach a higher degree of completeness, it could be investigated if CloudHost even makes sales outside of Europe, and if that's the case, locate the data about those sales and merge them with the data at hand.

To complete this discussion about completeness, it should be mentioned that Batini et al. go further and propose the concept of *completability* as a function over time that indicates how complete a dataset is at a given point in time [BS06, p. 27]. This is useful in scenarios where a dataset evolves over time, e. g., weather data from a sensor, or computer predictions of elections.

Missing Value Handling It is comparatively easy to detect the presence of missing values and measure their impact, e. g., by combining the intrinsic metadata types NULL count and blank count (cf. Section 4.2). However, it is much harder to reasonably decide how these values should be treated upon identification.

To illustrate this problem, consider how relational databases encourage the usage of the `NULL` marker for missing values. While this practice offers an elegant solution to unambiguously denote the absence of a data value, it still leaves room for interpretation of the reason why that may be the case in an individual instance. The inventor of the `NULL` marker, E. F. Codd, offered two possible meanings: "value at present unknown" and "property inapplicable". For example, in a table describing people, a column that stores email addresses may be `NULL` either because the respective person chose to not disclose their email address ("value unknown") or the person does not have an email address ("property inapplicable"). It can even be argued that a third meaning should be included: "not known if applicable" [BS06, p. 24]. In the people table, this would mean that, from a data point of view, it is neither known what the email address is, nor if it exists in the first place.

These three possible interpretations of `NULL` values are indistinguishable on the data level.

Thus, extrinsic sources are required to learn how each NULL value should be treated. This enables a very precise approach when data needs to be completed. In cases where a property is inapplicable it is perfectly acceptable to enter a NULL, and that should not be counted as a missing value, because it is, semantically speaking, the correct value. On the other hand, NULLs that represent unknown values should trigger an investigation with the aim to find out the true value. A similar investigation should be done in cases where it is not known if a property is applicable.

In some cases, NULL can simply be replaced by default values, e. g., a blank string or a zero. In the CloudHost sales example, the IT professional notices that there is a NULL value in the registration_fee column, denoted by a ⊥. The domain expert informs him that this is not the same as zero delivery charges, but rather indicates that the exact amount has not been determined yet and will be filled in at a later point in time. This circumstance is documented in the corresponding metadata, so that future data users are made aware of this fact. Furthermore, an exemption is made to exclude NULL values in this column from being counted in any data quality metric.

Data Age Data age is a measure of how old a piece data is, i. e., the difference between the current date and the date when the data has been generated or stored. It is related to data provenance, but, due to its importance, is considered here as a metadata type in its own right. Many different scenarios rely on data that is up-to-date and current enough to serve a specific use case. For example, when trading stocks, it is important to have the most current stock prices available to make informed decisions. Another example is weather forecasting, where the latest sensor readings are required to build accurate prediction models of how the weather is going to change.

Data age has also been referred to as the *currency factor* of data, and is regarded as one of two sub-dimensions of the data quality dimension *data freshness* by Bouzeghoub et al. They define currency as the "gap between the extraction of data from the sources and its delivery to the users" [Bou04, p. 60], which fits well into the context of the previous paragraph. The second sub-dimension of data freshness they propose is the *timeliness factor*, which "captures how often data changes or how often new data is created in a source" [Bou04, p. 60]. Thus, timeliness is not directly related to data age and will be considered later as part of the concept *update behavior*.

Working with outdated data is generally not desirable, and to avoid that one needs to be aware of the data age. Age is a metadata type that is applicable on every level of a dataset, i. e., each individual attribute value can have its own age, or groups of data values share the same age, or the whole dataset is of the same age. Depending on the implementation of the system that recorded or processed the data, some of these values may be already intrinsically available in the data. For example, in cases where the creation date of an entity is of importance, it is considered good practice to automatically add the current timestamp upon insertion into the database as an explicit attribute to that entity. This has been done in the CloudHost sales example, where the date column explicitly captures the day on which a sale took place.

There are different levels of granularity in which the data age can be measured, from nanoseconds to years. In scientific experiments, a very fine granularity and temporal resolution

may be necessary, whereas the sales example only records the date without a specific time of day. The required granularity of data age may even be dynamic within one scenario, depending on the considered time horizon. For example, stock prices need to be current to the minute (or even second) for regular day-to-day trading purposes. However, when looking at the same data retrospectively, e. g., the last ten years, then one stock price per day may be perfectly adequate for many kinds of analyses. In fact, it would be highly unreasonable to attempt to store historic stock prices for every second that has passed, because that would unnecessarily inflate the volume of the data and slow down analyses.

The data age is useful metadata in scenarios where the data is immutable, i. e., the data values are fixed upon creation and cannot be changed in the future. This is usually the case where the data expresses a recording of something that has happened in the past, like the stock exchange, weather forecasting and sales examples. In other scenarios, the data is mutable and can be changed at a later time. For example, people relocate their homes, so any dataset that stores addresses needs to be updated accordingly. Here, it makes more sense to consider the point in time when the last change occurred to a data value, instead of when it has been created.

Another category of use cases where knowing the data age can be highly beneficial is characterized by the fact that the dataset at hand is only a snapshot of an underlying data source. A snapshot is a full or partial copy of a data source that is taken at a specific point in time. Thus, creating a snapshot introduces a second notion of data age, namely the age of the snapshot, i. e., the time that has passed between its creation and the current date. Snapshots are frequently used when, for example, the data source is a live production database with restricted access due to performance or security concerns. Further, snapshots are useful to create data backups or maintain the history of the data source. Most systems that allow snapshot management automatically provide timestamps of their creation, making them easily accessible.

In the CloudHost sales example, it is learned that the snapshot date is `2018-05-09`. This enables several conclusions to be made. First, no sale that occurred after that date can be part of that dataset. If such a case should occur, it most likely indicates a data quality issue. Second, the month of May is not included completely. Thus, any aggregation by month needs to be interpreted accordingly. Lastly, if further data is added at a later point in time, only those sales after that date are necessary.

Data Validity Data is valid if it is applicable and usable in a given context. If the data has been recorded correctly and without errors, it can usually be assumed that it is valid. The most common reason for data to become invalid is that the real-world fact the data describes is changed, but the data is not updated accordingly. For example, when two people marry and one of them changes their last name from *Schomm* to *Schomm-von Auenmüller*, then every occurrence of *Schomm* is no longer a valid reference to that person's last name. Other reasons for the invalidity of data are inappropriate or faulty procedures that change the data into an illegal state. For example, a database query may deliberately disable foreign key checks and update some data rows, resulting in a violation of integrity constraints.

The validity of a data value can also be limited upon its creation. For example, if a Web

shop wants to offer a limited time promotion, it could do so by offering a coupon code that automatically expires after the intended time span. Other examples where auto-expiration is frequently used include account passwords and browser cookies. Many modern databases, especially key-value stores and in-memory databases, provide features for data expiration. The default way to handle expired data is to delete it. However, there are scenarios where invalid data should be preserved, e. g., for historical reasons, like in data warehouses.

In a data warehouse, old data is usually not deleted. Instead, timestamps are used to explicitly mark rows as being valid within a specific time interval. The easiest way to achieve this is by introducing two attributes `valid_from` and `valid_until` and set their values accordingly. This enables queries that only consider valid entries while still preserving old values that can be used to reconstruct specific points in time or other historic reports. The intricacies of such procedures are described by `Kimball` using the term "slowly changing dimensions" [KR13, p. 53]. Another approach to this issue is offered by so-called *temporal databases* [Sno86], which offer specialized concepts (e. g., *Valid Time* and *Transaction Time*) to deal with time-varying data in a more consistent manner.

In cases where the data validity is explicitly stored, it can be argued that it is no longer an *extrinsic* metadata type, because it is possible to derive it from the data itself. Thus, data validity is a concept that can either be intrinsic or extrinsic, depending on the use case at hand.

Update Behavior Most data sources are not static, but are changed and updated throughout their lifetime. When a dataset is updated, the results that have been computed with that data as input can become invalid, including its data profile, and their recomputation may become necessary. Thus, it is useful to document the expected frequency with which a dataset is updated, as that influences key design decisions down the line. For example, when a dataset is static and never or very rarely updated, then there is no need for automated data processing routines. In these cases it is most likely the most time-efficient and economic way to process such a dataset manually. On the other hand, if updates occur with a certain regularity (e. g., once a day, week, or month) then the initial costs for setting up automated processing will quickly reach the break-even point. ETL processes are prime examples for this type of automation. Lastly, the update frequency could theoretically also approach real-time, i. e., changes are reflected immediately and a reaction is needed as soon as possible. This requires a switch from batch to stream processing, which usually necessitates specialized real-time streaming technology.

Bouzeghoub et al. use the term *change frequency* to describe this concept and propose a classification with three categories: stable data, long-term-changing data, and frequently-changing data [Bou04]. This classification matches the previous considerations. However, it should be noted that the update frequency in reality is more like a spectrum that ranges from *never* to *continuously*, and any interval in between is possible.

Another aspect of the update behavior has already been mentioned earlier: the mutability of the data. Some databases only allow additional entries to be made to a dataset, and existing entries may not be changed. These databases are referred to as *append-only databases*.

In the CloudHost sales example, a customer can make a purchase at any time, and the

associated data is recorded with minimal delay. However, the data is not accessible in real-time. Instead, a snapshot is generated once a week. With this regularity in mind, the IT professional decides to set up an automated ETL process for data loading that is scheduled to run as a weekly batch job. Additionally, the data is immutable, so he does not need to worry about updating any entries that have already been loaded into the data warehouse.

Data Access Frequency Data needs to be accessible to be useful. However, not all parts of a database are accessed equally often. Usually, the most recent data that is put into a system is the most relevant and thus, retrieved most frequently. When the volume of the data reaches non-trivial amounts, i. e., Big Data solutions are necessary, this can lead to performance bottlenecks if handled improperly.

This issue can be tackled with a data storage strategy that considers the data access frequency of the various parts of the data and attempts to strike an appropriate balance between performance and costs. One such strategy is described by Teradata Corporation, an enterprise software company that develops and sells databases [36]. They use the analogy of *temperature* to describe a spectrum from hot to cold [Gra12] and recommend appropriate storage technology that ranges from high-performance, high-cost to low-performance, low-cost. In this spectrum, five different tiers are identified:

- *Blazing* is the highest tier. This data should be kept in main memory at all times, because it is accessed the most frequent.
- *Hot* is the next tier, for which solid state disks (SSDs) are a good fit.
- *Warm* data lies in between hot and cold. Reasonably fast hard disk drives (HDDs), i. e., with at least 10,000 RPM, offer a good balance between bandwidth and cost for this tier.
- *Cold* is the tier that is best supported by economical storage, like high-capacity HDDs with less than 7,500 RPM.
- *Arctic* as the lowest tier is the place for data that is only very rarely touched, if ever, but still needs to be stored, e. g., for regulatory purposes. Historically, the storage solution of choice for this tier has been magnetic tapes. These have slowly been superseded by low-cost HDDs in recent years.

Depending on the data volume and the available budget, one might be tempted to resort to the KIWI principle, which is short for “kill it with iron”. Under this principle, any classifications of data tiers are skipped in favor of simply throwing as much money and hardware at the problem until it is solved sufficiently, i. e., buy as much RAM as is needed to store everything in memory. Such a strategy may be feasible in small-scale scenarios, but will quickly become unjustifiably expensive.

The data access frequency is a very dynamic type of metadata. The distinction between what is considered *blazing* and what is *hot* may change during the operation of the database. In fact, these classifications are rarely decided upon by humans. Instead, the data access is monitored live by a system and the classification and subsequent movement of data parts is performed automatically.

Table 6.3: List of extrinsic metadata types.

Metadata	Description
Data Provenance	Documents where the data comes from (machine-generated vs human-generated data) and how it has been transformed so far. Also known as data lineage or data pedigree
Data Privileges	Governance of access rights to a dataset
Data Location	Description of where the data is located at. Can be given in form of a URL
Data Completeness	Measure of how many of the expected entities and attributes are present in a dataset
Missing Value Handling	Documentation of how missing values, such as `NULL`s, should be handled
Data Age	Measure of how old the data is, i. e., the difference between now and its time of creation
Data Validity	Binary classification whether a piece of data is valid or not in a given context
Update Behavior	Assessment of how often data is updated, and if so, how
Data Access Frequency	Assessment of how often data is accessed and read

Summary A summary of the preceding extrinsic metadata types is given in Table 6.3.

6.4 Sources of Extrinsic Metadata and Their Extraction

The core characteristic of extrinsic metadata is that it cannot be derived solely from a given dataset. Thus, new sources are necessary with new approaches to retrieve such metadata. This section gives an overview of where extrinsic metadata can be found and how it can be extracted.

Domain Experts The people that deal with datasets on a daily basis are likely to be highly familiar with it. Maydanchik states that "nobody knows the data better than the users." [May07, p. 152]. This means that if they can be asked about a specific dataset, extrinsic information can be learned. In the sales example, it was the IT professional that could pose questions to the domain expert (cf. Figure 6.1). There are three basic requirements that must be fulfilled for this approach to work.

First, it must be known whom to ask. Within a small company, this should be fairly straightforward, whereas it gets progressively harder in larger settings. If the dataset has been sourced externally from, e. g., the Internet, it could even be entirely unfeasible to attempt to find a person that has expert knowledge about it.

Second, the person in question needs to be reachable via a channel of communication. These range from personally going over and interacting with them, over using voice chat or a telephone, to asynchronous media like text message or email. Which of these channels

should be used depends on factors like the distance between the two parties and their relation. For example, if the other person is a colleague working in the same building, it makes the most sense to have a face-to-face meeting. However, if the other person is a stranger with an unknown affiliation, the best approach could be to write a polite email and ask nicely.

Third, the contacted person must be willing to cooperate. Within the same company this should be easy to accomplish, but there are many reasons why somebody might decline such a request. These include, e. g., lack of time or motivation, too little confidence in one's own knowledge, or simply a dislike towards the other party. This issue is exacerbated when the contacted person is not working for the same company as the inquirer as that may diminish any motivation to cooperate and help out.

When all three requirements are met, a transfer can begin in which the domain expert discloses his implicit knowledge about the dataset. This transfer can assume the form of an interview, in which a set of questions that refer to the extrinsic metadata types are asked and answered. The results should be recorded and stored in a metadata repository. In some cases, the interviewee may not know certain answers, but he can name somebody else that might know it. This leads to a recursive process in which the interviewer goes from one person to another in a pursuit of information, until he has finally learned everything he needs to learn about the data. In the context of data quality assessment, Maydanchik proclaims that "often it is simply useful to sit next to those data experts and watch them while they work." [May07, p. 152]. While this may be a bit too intrusive in some settings, e. g., when customers are directly involved, like in a bank or at the dentist, it is certainly a good approach where it is feasible and the affected data experts do not mind the additional attention.

Alternatively, instead of an interview, the questions could also be asked in form of a check list or survey that is sent to the domain experts, asking them to fill it out. Such a check list could, e. g., be derived from the list of extrinsic metadata types as summarized in 6.3. This has the benefit of being asynchronous, i. e., there is no need of setting up any kind of meeting and the domain expert can think carefully of his answers on his own time. Additionally, it allows the IT professional to simultaneously pose questions to a larger group of people. For example, if the dataset in question is composed of data from various departments across a company, then there are probably different people who know about the provenance of individual parts, their respective schema completeness and validity, and how that data is updated. Asking multiple people from these different departments at once has the potential to significantly speed up the extraction of such extrinsic metadata.

It must be pointed out that any information learned this way should be assessed and verified critically, as it may very well be the case that the question has been misunderstood, or the asked person does not know and just guessed, or even worse, intentionally spread false information. Double-checking during the early stages of a project can prevent a lot of issues that might be very costly to fix later on.

Explicit Recordings Some of the extrinsic metadata that is sought after might be available in the form of explicit recordings. Such recordings may be available, e. g., in written database documentations or visual models. Olson suggests to look for "procedure manuals used by data entry personnel" [Ols03, p. 125], as these may contain rules on how the resulting data

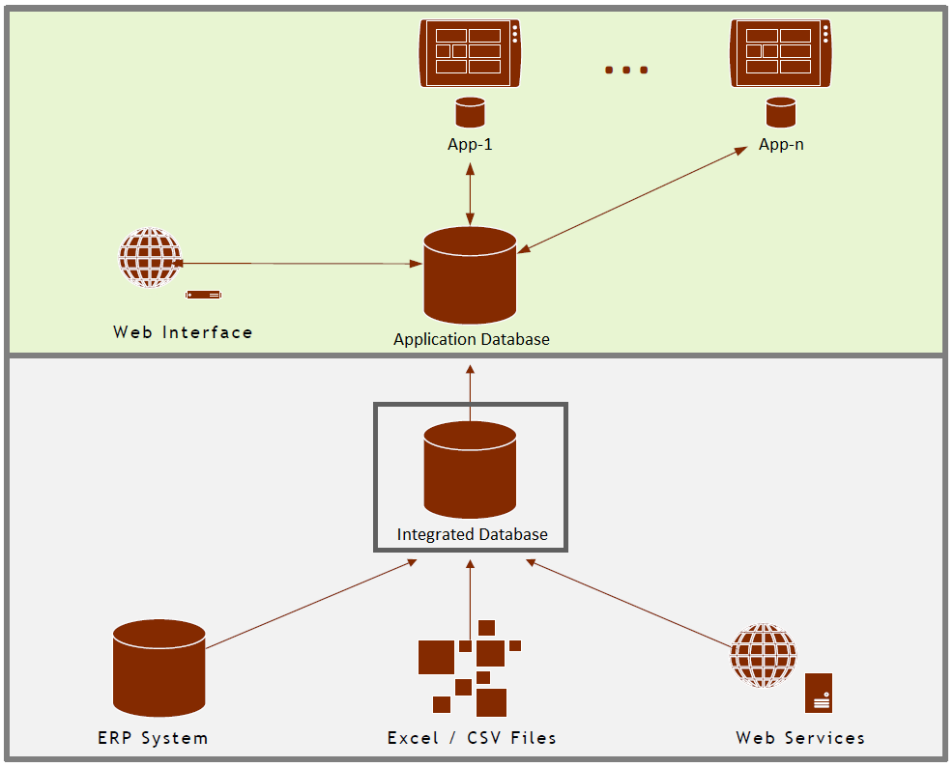

Figure 6.2: Model of the data flow in an exemplary project seminar.

will be structured and formatted.

Once relevant recordings have been identified, they need to be examined carefully in order to extract metadata. It helps when this task is approached with clearly formulated questions in mind, such as "who has the privileges to change the data" or "how often is the data updated". This focuses the attention and mitigates the risk of getting lost in a wealth of potentially unrelated information. Each piece of information that is discovered should be recorded in a metadata repository.

As an example of how explicit recordings can be used for the extraction of extrinsic metadata, consider the model shown in Figure 6.2. It is taken from the results of a project seminar that took place at the Department of Information Systems at the University of Münster during the summer of 2015[1]. It shows the relevant components of a system and the data flow between them.

At the bottom are three sources that are merged by means of ETL processes into an integrated database, which acts as a central point of truth. The data that arrives here is used to populate a separate application database, which is copied to a number of independent databases at remote locations. These are used to serve individual instances of a customer-facing application. Lastly, a Web interface is connected to the application database to allow data monitoring and manual changes.

[1]For compliance with non-disclosure agreements, names and references have been omitted. If needed, the complete report of the project seminar can be obtained from the author of this thesis.

Assuming that extrinsic metadata about the application database is required, there are several things that can be learned from this architecture and its documentation. First, it gives a good overview of the *data provenance*. The data sources are clearly identifiable and show where the data originates. To learn the exact transformation the data undergoes, the respective ETL processes can be investigated. Next, properties of the *data location* are revealed: the application database is copied to the physical location of the remote sites.

Information about the *update behavior* can be read directly from the ETL processes, which are executed daily (this is the update frequency) and perform exclusively append operations to the integrated database, which means that the data is immutable. On the other hand, the Web interface allows an inspection and change of specific data values in the application database, which gives the administrator an option to manually override any errors or data quality issues he encounters. However, from the directions of the arrows it can be seen that these changes are not relayed back to the integrated database or the sources.

Lastly, the system architecture allows to draw inferences about the expected *data access frequency*. The read load on the application database scales with the number of individual applications that are being set up, while the load on the integrated database is unaffected by that.

Application The third source for extrinsic metadata considered here is the set of applications or programs that read or process the dataset in question. After all, many types of data are the direct result of the execution of an application, so inspecting that source can reveal interesting facts about the data. The extent to which this is possible depends on the accessibility of the concerned application. It can range from *no access*, e. g., because it is unknown or proprietary, to *full access*, e. g., the source code is available for inspection. If there is no access at all, nothing can be done here, and the extrinsic metadata must be obtained from one of the other sources. In most cases however, there should be at least some trace that allows inferences to be made. In cases where the data is exchanged through an interface, this interface can be examined. Olson recommends to "seek out interface definitions to application programs that feed data to the data source. These can often add illuminating information about the expected content of the data." [Ols03, p. 125].

For example, a popular way to share data is through a REST API. REST stands for "representational state transfer" and was designed as an architectural style for Web services and interfaces [Fie00]. It is considered good style to properly document a REST API [Mas11, p. 55], i. e., add written information about the functions that are offered. Among others, these information include allowed request types such as GET, POST, or DELETE, which correspond to the *data privileges* a user has. Furthermore, an API documentation can also reveal intrinsic metadata, such as the data type or format of the offered data.

If the data is not exchanged via an API, but instead directly created by an application running within the same control sphere (e. g., the same organization), it is advisable to directly examine the application. Ideally, the respective source code is accessible and can be examined for clues about the resulting data. Olson refers to this practice as "source code scavenging" [Ols03, p. 221] and recommends it as a way to gather simple and complex data rules (cf. Figure 4.3). Inspecting the source code of an application can be an extensive task, because it

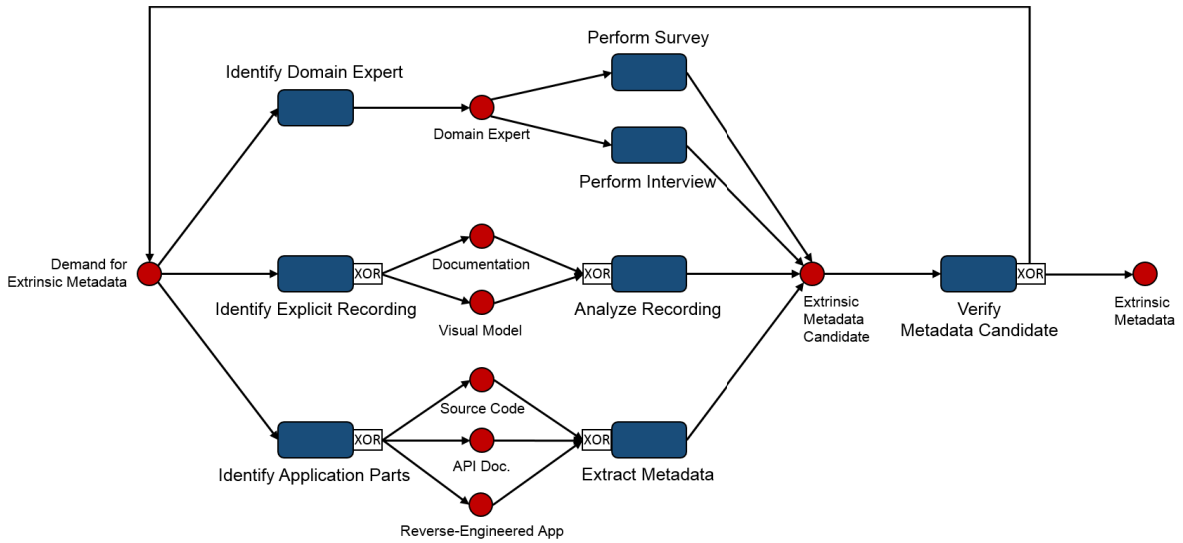

Figure 6.3: Extrinsic metadata extraction process.

can quickly reach hundreds of thousands of lines. Still, it is usually sufficient to only examine those parts of the code that interact with the data, e. g., serialization routines or prepared statements. This enables drawing conclusions about *data provenance* and *schema completeness*.

Finally, there may also be scenarios in which neither a documented interface nor source code is available, and the application is essentially a black box. If it is of mission-critical importance to learn more about the data creation mechanisms of such an application, it may be worthwhile to attempt to reverse engineer it [EC05]. In this process, the application is deconstructed into its individual components with careful observation how each of these behaves. From these observations, the application design and blueprint are derived, which may contain useful information about the dataset. Another form of reverse engineering is *redocumentation*, where the intent is to recover lost or non-existent documentation about a system [CC90, p. 15]. In any case, reverse engineering is usually a costly and time-consuming process which should only be resorted to if no other means for gathering information about extrinsic metadata are available.

This section dealt with ways in which an application can be inspected to extract extrinsic metadata. However, the application itself can also be profiled on its own in order to derive application metadata. This will be discussed later in 7.2.2 under the name *application profiling*.

Extrinsic Metadata Extraction Process The various sources for extrinsic metadata and means to extract them are brought together in one concise process model, as depicted in Figure 6.3. Beginning with the demand for extrinsic metadata, one of the three paths is selected and a respective element identified. All paths lead back to an extrinsic metadata candidate, which is verified before it is considered applicable. If the verification step fails, the process is started again from the beginning where another source can be chosen.

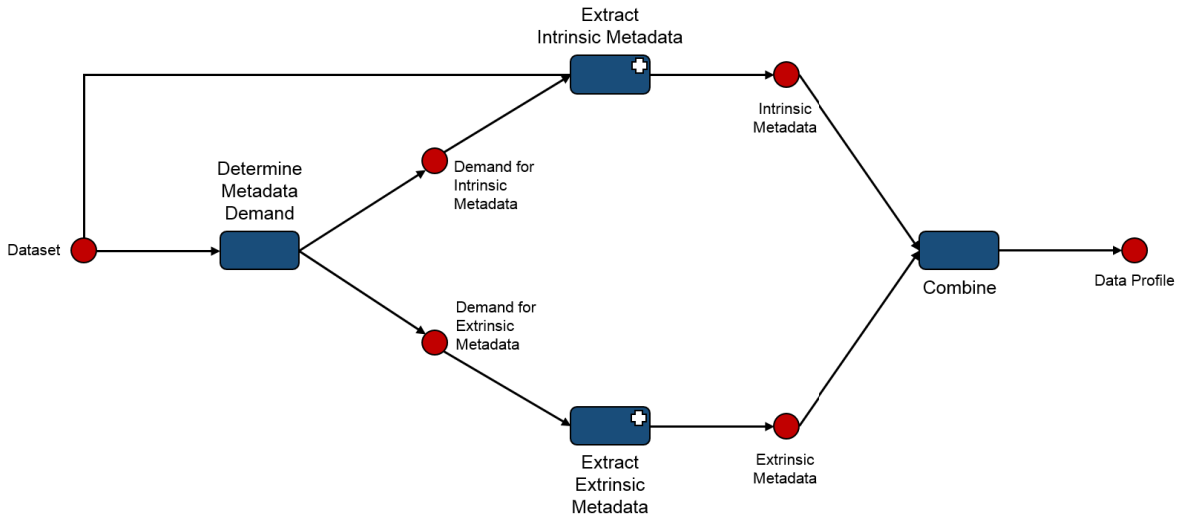

Figure 6.4: Combined metadata extraction process.

6.5 Bringing Intrinsic and Extrinsic Metadata Together

With the process to extract intrinsic metadata (cf. Figure 4.4) and the process to extract extrinsic metadata (cf. Figure 6.3), it is now possible to construct a super process that combines both processes into one combined process. This combined process is shown in Figure 6.4.

The process begins on the left-hand side with a dataset, for which a specific metadata demand is determined. This demand can now pertain to either intrinsic or extrinsic metadata, and is subsequently passed on to the respective extraction step. These extraction steps are marked with a white plus symbol in the top right corner to indicate the link to the respective sub processes, as shown before in Figure 4.4 and Figure 6.3. The results from both the intrinsic and extrinsic extraction processes are then combined to form the resulting data profile.

Such a data profile is a set that contains all the extracted metadata and their respective values. A conceptual overview is given in Figure 6.5, which combines the findings of Tables 4.3 (single field metadata types), 4.5 (multi field metadata types) and 6.3 (extrinsic metadata types) in one hierarchical presentation.

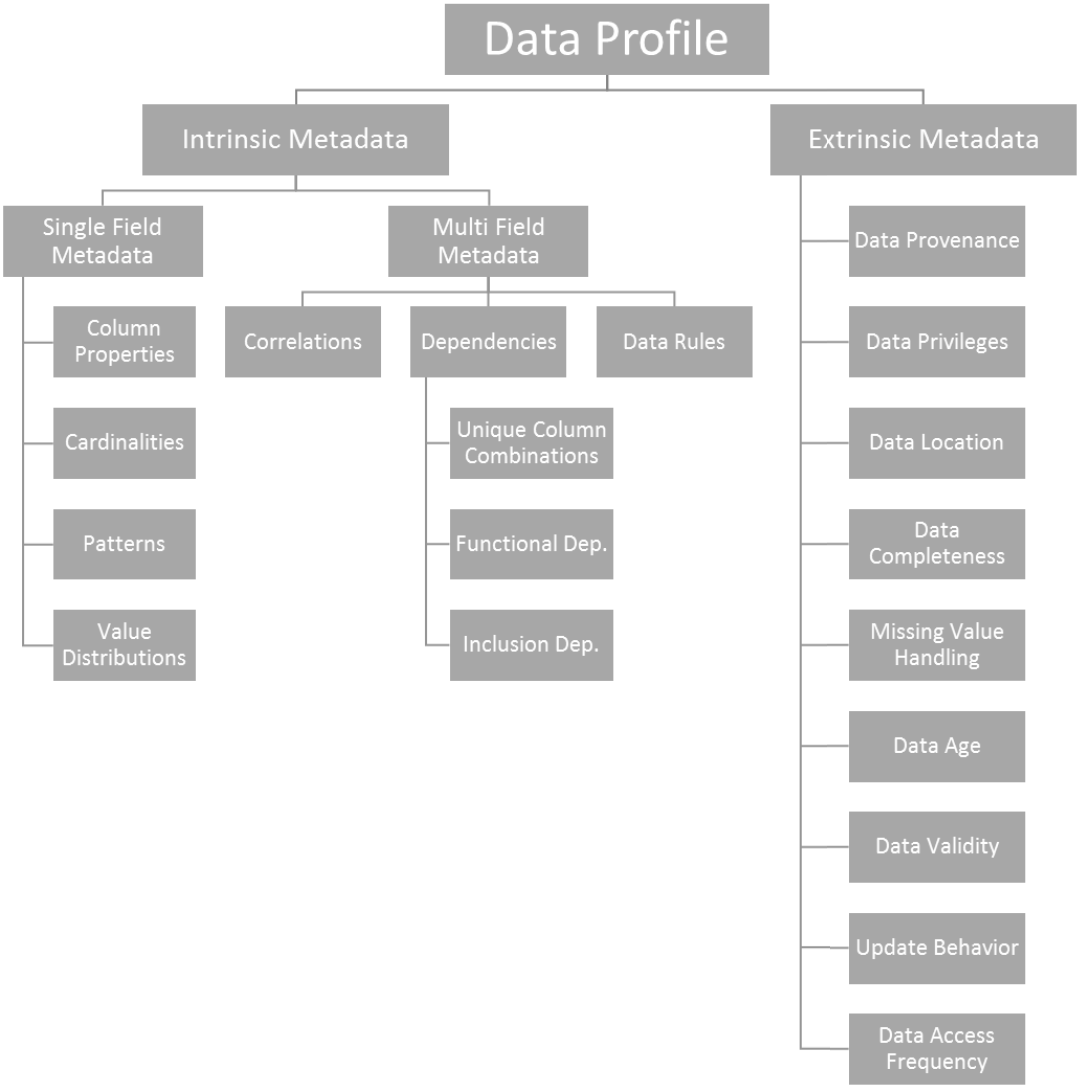

Figure 6.5: Conceptual overview of all discussed data profile types.

7 Profiling More Than Data

Performing data profiling is a recommended first step when facing an unfamiliar dataset or new data sources. However, looking only at the data may not reveal everything there is to know. Chapter 6 introduced the notion of extrinsic metadata, which covers the "other side" of the data that is not explicitly observable. Still, this is a data-centric perspective that does not fully capture the reality that most organizations face. As a matter of fact, data is not always the focal point and rarely exists in an isolated vacuum. Instead, it is usually only one component of a larger, surrounding context, which consists of the systems that interact with and depend on the data. This context can contain interesting pieces of information that may be highly beneficial in a data-driven project. Thus, a change in perspective is proposed here that shifts the focus away from the data as the central point of attention, towards a more holistic view that incorporates the context of the data.

To achieve this goal, an abstract representation of the context of data is needed in order to enable reasoning about it and provide a framework for discussion. This is addressed in Section 7.1, which introduces the Generic Application Framework (GAF) and its levels. Next, Section 7.2 describes how the general idea of profiling, i. e., observing and analyzing an object to construct a concise summary, can be extended to each individual level of the GAF. Lastly, a set of three approaches is established that allows an operationalization of the framework and makes it applicable to real scenarios in Section 7.3.

7.1 The Generic Application Framework

The Generic Application Framework (GAF) is a simple model that consists of three levels: *application* at the top, *process* in the middle, and *data* at the bottom (cf. Figure 7.1).

The bottom-up relationship between the individual levels is that a lower level *enables* the higher level, i. e., data enables processes, and processes enable applications. This perspective is supported by Weske, who states that "data are an integral part of business processes." [Wes12, p. 294]. In a top-down reading direction, the relationship is such that a higher levels *uses* the next lower level, i. e., an application uses processes, and processes use data. This relationship is not strictly required, and there may very well be datasets or processes that are used by multiple processes and applications, respectively. In this sense, the GAF is only one view on a specific part of components within an organization.

The GAF is intended to be used as a template onto which a wide variety of process-oriented and data-driven systems can be mapped. For example, a bank may have a credit card management system in use, which represents an application in this context. This application consists of many processes, e. g., for issuing new credit cards or processing transactions. These processes require specific data sources to be available, like a customer database and a transaction

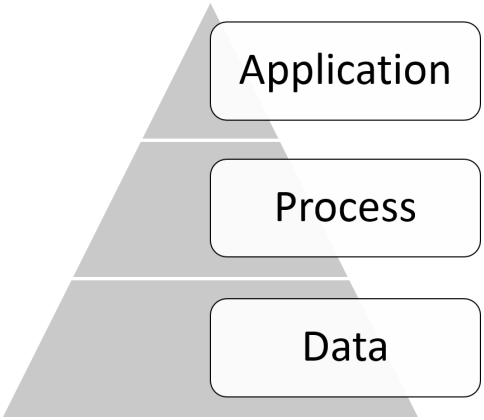

Figure 7.1: Generic Application Framework.

database. Many more systems that revolve around processes and data can be mapped onto the GAF in a similar fashion, including broad application classes such as enterprise resource planning (ERP) systems, data warehouses, or customer relation management (CRM) systems. This makes the GAF widely applicable to numerous different types of use cases.

7.2 Profiling the GAF Levels

Each level of the GAF can be profiled individually. In other words, there are three associated procedures, namely data profiling, process profiling, and application profiling. The first was described extensively in the previous chapters, with an extensive overview of the state-of-the-art in Chapter 4, a list of variants in Chapter 5 and an extension into extrinsic metadata in Chapter 6. All these different metadata types, methods, algorithms, tools, and concepts are applicable here.

The other two forms of profiling have not been considered so far. Thus, they are described in dedicated sections next.

7.2.1 Process Profiling

A process is defined as a "completely closed, timely and logical sequence of activities which are required to work on a process-oriented business object" [BKR14, p. 4]. Furthermore, the term *business process* is used to describe a subclass of processes that are used by businesses and organizations to realize their goals [Wes12, p. 5]. The remainder of this section focuses on business processes and will use the simple term *process* to refer to them.

Profiling a process is an activity that involves an analysis of the process with the goal to extract and record descriptive metadata about it. The types of metadata that can be extracted from a process depend on the type of process and how it is defined. Elias et al. propose a business process metadata model (BPMM) [ESJ10], which lists a number of metadata types for business processes. Although the primary use case described by the authors is a facilitation of searching and navigating in a process repository, their model includes numerous metadata

types that capture important characteristics of business processes. More specifically, the individual process metadata types are the following:

Process Area Based on the Porter Value Chain [Por85], the process area is a classification attribute that denotes the area within an organization to which a process belongs. There are primary process areas (i. e., inbound logistics, operations, outbound logistics, marketing & sales, and service & mainteance) and supporting process areas (i. e., procurement, technology development, human resource management, and firm infrastructure).

Process Phase The phase of a process shows to which set of operational activities a process belongs. The proposed classification scheme by Elias et al. consists of five phases: planning, identification, negotiation, actualization, and post-actualization.

Process Type The process type classifies how a process interacts with its resources. It can be either *exchange* (i. e., a resource is received or given from or to another party) or *conversion* (i. e., a resource is used or consumed and coverted into something else).

Process Relationship A process can have relationships with other processes. Applicable relationship types are *generalization/specialization* and *partof/includes*.

Process Level The level of a process refers to the organizational level within a company on which the process is executed. It can be either operational, tactical, or strategic.

Resource A process consumes or produces resources, which can be classified into one of these categories: goods, services, rights, finances, and information. In particular, the required datasets and sources are considered here.

Actor This is the person responsible for the execution of the process. Actor categories are: customer, supplier, employee, investor, and organizational unit.

Business Context The context in which a process is executed is captured here. It contains information that relate to the industry, the involved communication channel, geopolitical descriptions, and official constraints such as legal or regulatory requirements.

Goal This is a description of a condition or state of affairs that an actors seeks to achieve. Goals can either be *soft goals*, i. e., abstract, strategic objectives that are strived towards, or *hard goals*, i. e., operational goals that must be reached by the process in order to succeed.

Extracting these types of metadata from a process is likely to prove difficult, because they depend on the intentions of the process designer. Thus, extrinsic sources, like the knowledge of a domain expert or written documentations, are necessary here, similar to the extraction of extrinsic metadata (cf. Section 6.4).

If the process in question is available in form of a process model, e. g., as a Petri net, then new possibilities for process profiling arise. For example, the quality of a process model can be measured with the use of quality metrics. These metrics aim to evaluate the process model with regards to one specific characteristic, such as the number of edge crossings or the

average connector degree. There exist numerous different process model frameworks that define various such metrics. A recent work of Pflanzl contains a comprehensive overview of this topic and concludes with a framework that defines quality metrics in the following categories [Pfl17, p. 54]:

Planar Variables capture characteristics of the representation of the process model in a plane, i. e., a two-dimensional space. Metrics of this category include the number of edge crossings, the number of edge bends, node occlusion, crossing resolution, angular resolution, consistent flow direction, and orthogonality.

Retinal Variables relate to the use of graphical elements, e. g., colors, shapes and textures, in a process model. These can be used to realize concepts such as syntax highlighting, label styles, or activity icons.

Complexity is an inherent property of a process model that is independent of how it is represented graphically. It can be assessed in multiple different dimensions, including size measures (e. g., the number of elements), connection measures (e. g., the density of connectors), modularity measures (e. g., the separability of the process graph), connector interplay measures (e. g., the connector mismatch), and complex behavior measures (e. g., cyclicity).

Textual Contents in a process model are used to label elements and provide descriptions. The quality of these can be measured in metrics such as naming convention adherence or text complexity.

Semantics relate to the question whether the process model is a correct representation of the corresponding domain. Metrics in this category pertain to dimensions such as completeness and validity.

For many of these metrics, concrete algorithms can be specified that define precisely how it is extracted from the underlying process model, which allows an automation of their computation. For example, the planar and retinal variables, as well as the complexity, are easy to assess by an appropriate software tool. Similarly to the definition of intrinsic metadata in the context of data profiling, the metrics can be considered to be intrinsic metadata of the process profile. However, some of the metrics cannot be assessed automatically, like the completeness or validity of a process model. These require the knowledge of a domain expert, analogical to the extrinsic metadata of data profiles.

Relation to Process Mining In the context of process profiling, it makes sense to describe and compare a related discipline called *process mining*. Van der Aalst writes that "the idea of process mining is to discover, monitor and improve real processes (i. e., not assumed processes) by extracting knowledge from event logs readily available in today's systems." [vdA11, p. 8]. Thus, there are three different main types of process mining: discovery, conformance (monitor), and enhancement (improve). All three types require an event log, i. e., a chronological list of events observed by a system, as their input. For discovery, this event log is the only input,

and the result is a reconstructed process model. The algorithm used for this transformation is called α-algorithm and was invented by van der Aalst et al. [vdAWM04]. The other two types of process mining, conformance and enhancement, take a process model as an additional input. The goal here is to compare the executed process (as observed in the event log) with the formally defined and designed process model, in order to then verify that these two correspond to each other (conformance), or to suggest improvements to the process model (enhancement).

In contrast to this, process profiling has only a single input, namely the process or its model, which is used to extract metadata about it. The goal is to provide a user with a concise view on the most important characteristics of the process in order to facilitate his understanding of it.

7.2.2 Application Profiling

The term *application* is used to describe a broad spectrum of computer programs, ranging from small special-purpose tools (e. g., smartphone apps that show the weather) over desktop software (e. g., word or spreadsheet processors) to distributed systems that span whole organizations (e. g., data warehouses or management information systems). Most instances of these various applications can be meaningfully interpreted as consisting of a collection of processes. These are the applications that are considered in the context of the GAF, and they share many similarities with information systems. For example, the credit card management system mentioned earlier is an application that consists of processes, and thus, is applicable for the GAF.

The concept *application* is not as well researched as the other two concepts of the GAF, *process* and *data*. Thus, there is little prior work in the area of application profiling. In fact, some people have a completely different understanding of the term *application profiling* and use it to describe the practice of profiling a running piece of software [12][30]. However, for the scope of this thesis, this was described as *software profiling* in Chapter 2. Thus, there is a certain degree of ambiguity that needs to be taken into account when researching this term.

Here, application profiling is defined as the extraction of metadata that capture the properties of an application. This yields a general overview of the application, its purpose, and the environment and resources that are needed for its operation.

In order to perform application profiling, a list of metadata types is required, so that the user knows which information he needs to gather. Seeing how application profiling is not yet an established discipline, such a list cannot be found in the literature. To address and fill this gap, the rest of this section aims to establish a first proposal on what this list of application metadata types should contain.

Application Type Applications can be broadly classified according to their type. Research on information systems has identified the following types, which are also applicable here [LL17]: transaction processing systems [GR93], management information systems, decision support systems [SWC+02], and executive support systems [RD88]. Further application types are search engines [CMS10], data warehouses [KR13], geographic information systems [HCC11], supply chain management systems [KH13], customer

relationship management (CRM) systems [BM15], enterprise resource planning (ERP) systems [O'L00] and knowledge management systems [AL01].

Knowledge about the type of an application is useful for assessing its purpose and its role in the larger context of the organizational IT landscape. Note that this classification is not meant to be strict, i. e., some of these types may overlap. For example, one application may be considered a transaction processing system and a customer relationship management system at the same time. Furthermore, this list is a representation of some of the most popular application types, but does not claim exhaustiveness. Thus, the attention here is restricted to these types. Depending on the required level of granularity, it may also make sense to further subdivide the individual types.

Users The users are the people that use the application. These can be specified either by role, e. g., "middle management", by organizational affiliation, e. g., "members of the sales department", or directly by name, e. g., "Mr. Smith". When the group of users of an application is known, it can be inferred who might be a domain expert, and who is likely to be affected by any changes to that application.

Depending on the application, users may also be subdivided into user groups that use the application in various ways. For example, a data warehouse may have different people that are assigned with tasks such as data input, data administration, and report generation. Each of these tasks requires individual privileges within the application.

Application Processes These are the processes of which the application consists. Ideally, these are fully documented and described with their name and profile, as described in the previous section.

Resources An application needs resources for its successful execution, either as input, output, or intermediate byproduct. Most of these resources are referenced implicitly through the included processes, but for some applications a declaration of further resources may be necessary, such as the context, laws, or other regulations. Categories of resources include goods, services, rights, finances and information/data.

Vendor The vendor provides the software that is used to run the application. This can either be a third party, i. e., a software company that implements and offers software to others, or the organization itself, which is called an in-house implementation. Both approaches have advantages and disadvantages that must be evaluated in each individual use case [IJ90].

Technical Context The technical context captures where an application is executed, which can either be local or remote. If it is remote, further distinctions can be made according to whether the application is executed in an on-site data center, or using a cloud-hosting platform. Further information about the technical context may include any requirements a machine needs to meet, e. g., memory size or CPU speed, as well as dependencies on other software components, such as the operating system or programming libraries.

Business Context Similar to the business context of a process as outlined previously, the business context of an application contains information about the surroundings in which it is executed. This relates to the industry of the organization, geopolitical descriptions, and official constraints such as legal or regulatory requirements.

Purpose The purpose is a high-level description of why an application is needed and which types of activities it enables and serves. Such a description will most likely reference the other metadata types, such as the users and processes.

Gathering these various types of metadata types is a complex task that is likely to be not automatable in any meaningful way. Instead, it is a manual process in which information is collected from various sources and brought together in an application profile. Such a profile is useful for a concise summary of the most important properties of a given application and can be used for purposes such as comparison, evaluation or search of applications within an organization.

7.3 Proposed Appproaches

The first application of the GAF is to decompose a use case into the three levels and profile them individually, as shown in the previous section. However, the true potential of the GAF is unfolded when it is fully embraced and used to move between the levels. In order to do that, a starting point must be declared, which may be any of the three levels. Consequently, three different approaches are proposed here, namely the data-first approach, the application-first approach, and the process-first approach. These approaches correspond to a bottom-up, top-down, and inside-out strategy respectively, and each is described in a dedicated section next.

7.3.1 Data-First Approach

The data-first approach assumes that a dataset is available from the beginning. From this starting point, the other levels are reached in a bottom-up fashion, as shown in Figure 7.2.

It consists of the following steps:

1. Profile the data First, the dataset is profiled according to the data-oriented data profiling variant described in Section 5.1. Additionally, the resulting data profile may be augmented with extrinsic metadata, as explained in Section 6.4. This provides an overview of the general state of the data and its quality, which are relevant for the next step.

2. Identify enabled processes This step aims to answer the question "which processes are enabled by this data?". This is based on the assumption that data is needed for the successful execution of a process. For example, a company may have a process *send newsletter* that is executed regularly to send out newsletters to their customers. This process requires address data of all intended recipients. Thus, a dataset that includes customer addresses is needed to enable this process.

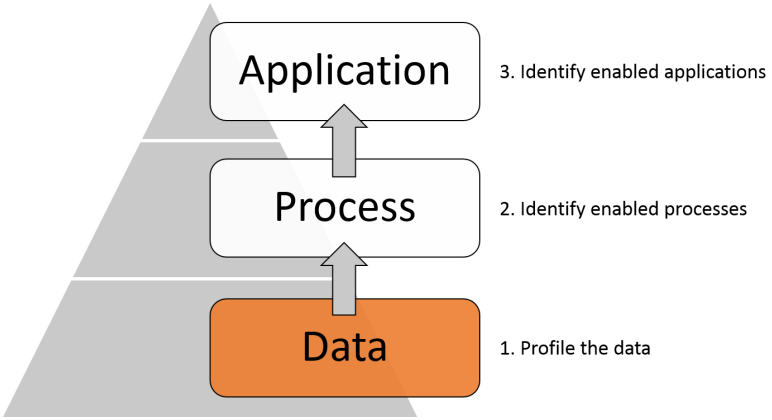

Figure 7.2: Data-first approach.

Finding all processes that are enabled by the data can be a quite extensive task. One approach to facilitate this is using a process repository, if one exists. A process repository is a central place in which all process models of an organization are stored and made accessible [MWA+07]. It provides a search space in which the dataset at hand can be compared against the input definitions of established processes. Another approach is to look at reference process models, which can be obtained from the literature or public repositories. This allows to find processes that are not yet implemented within an organization.

If the available data represents an event log [vdA11, p. 104], then it can be used to perform process discovery, i. e., reconstruct a process model from the data.

3. Identify enabled applications The set of processes that are enabled by the data finally allow to draw conclusions about any enabled applications. These can either be applications that are already implemented at the organization, or novel applications that have not been considered before.

The CloudHost Case The data-first approach is demonstrated in the CloudHost case, as introduced back in Section 1.3.

Remember that CloudHost operates as a platform provider that hosts software products for its clients. CloudHost records a wide variety of data about which client is running which software, in which version and configuration. This data lead the managers to pose the following question: “How can we use this data to optimize the work of our sales team, and provide them with custom-tailored client-specific sales arguments?”. With this question and the data, CloudHost approached a team of IT consultants and initiated a project to find an answer. As this project revolves around a specific dataset at hand, which is readily available from the start, it is a good example for the application of the data-first method.

The first step taken was to thoroughly profile the data, which was provided as a CSV file. This revealed that the data contained 80 columns and 46,548 rows, as well as a header row containing column names. For this volume of data, a relational database was deemed sufficient, and the decision was made to use MySQL for data storage. The columns included a client ID, a software name, a version number, numerous attributes regarding the state of

the software, as well as multiple columns whose names were not self-explanatory. In search for a primary key, it was found out that no single column is entirely unique. Thus, a search for unique column combinations was started, which revealed that a combination of client ID, software name and version number yields a key candidate, which was consequently promoted to a primary key. This allowed the creation of a schema for the database that was populated by means of an ETL process.

Through interviews with domain experts it became apparent that the provided dataset did not include all relevant data to assist the sales team. In particular, further information about the clients, such as name and total revenue, as well as further information about the software was needed, which led to the conclusion that the *schema completeness* (cf. Section 6.3) was lacking. This issue could be remedied by requisitioning the data in question and integrating it into the database.

After the data was profiled and properly integrated in one database, the identification of enabled processes as the next step could commence. An observation was made that the data included information about which software products are good supplements to other products. This led to the idea of exploiting a technique called *cross-selling* [KWdRM03], i. e., selling additional products to an existing customer based on the products that he already has. For example, product A and product B are complementary to each other. In cases where a customer has purchased product A, but not product B, a potential sale can be made by recommending product B to him. Subsequently, a process was implemented that enabled cross-selling by performing the necessary data analysis steps and providing the resulting recommendations to the sales team.

Using the same rationale, additional processes were identified and implemented. For example, the CloudHost executives stated that they would like more customers to use their in-house software instead of third-party products, because this resulted in more net profits. This was addressed in a process that analyzed the data for cases in which existing third-party software could be replaced by functionally equivalent CloudHost products. Similarly, a process to recommend an update of outdated products, based on their version number, was set up.

With all these different processes, the last missing piece was a unified interface that enables the head of the sales team and other users to see what is happening. Thus, an application, called the *CloudHost sales management cockpit*, was set up that provided a Web interface where users could log in and oversee the processes, adjust parameters, execute calculations, and inspect the results. Additionally, the individual recommendations made by each process could be weighted and ranked, so that a prioritization of sales activities was made possible.

This concludes the example of the data-first approach, where it was shown how the GAF can be used in a bottom-up fashion, starting with a profiling run of the available data, identifying enabled processes, and finally working up to the application level.

7.3.2 Application-First Approach

In the application-first approach, the starting point is the top level, i. e., the application. From there, the procedure continues downwards in a top-down fashion, as shown in Figure 7.3.

Scenarios in which this approach can be meaningfully used are usually characterized by a particular requirement that can be met with a specific application. For example, a manager

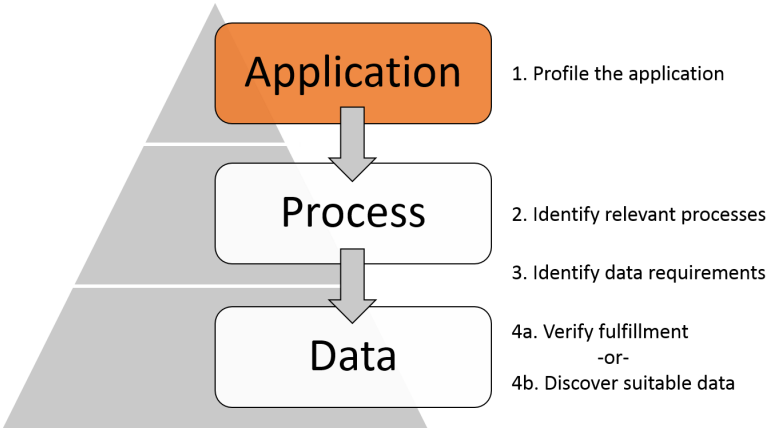

Figure 7.3: Application-first approach.

might say "we need to keep better track of the materials in our production facilities!", which leads to the initiation of a project to implement an appropriate application. Such a project can use the GAF with the application-first approach for the initial phases and requirements elicitation.

Another type of use case occurs when an application is already implemented within an organization, but its performance is insufficient. The GAF can be used then to assess the application, analyze its context and identify potential deficiencies

The application-first approach consists of the following steps:

1. Profile the application The first step is the creation of an application profile as described in Section 7.2.2. This includes an investigation into the different application metadata types and their values, which should provide a good first overview of the characteristics of the application. Following the list of metadata types ensures that no important aspect is left out and encourages the responsible people to carefully think about the application from every relevant perspective. Furthermore, the compilation of all this information in one place, and a subsequent communication to all involved stakeholders, ensures that everybody is "on the same page".

In the example, a call for better tracking of production facilities may be addressed by an ERP system. Thus, this application type should be selected accordingly and recorded as part of the application profile. In particular, the individual processes that are involved in the application must be identified, which leads to the next step.

2. Identify relevant processes After the application has been fully profiled, the relevant processes need to be identified. "Relevant" here means that the process is necessary or useful in the context of the application, i. e., it is needed for its successful execution. One technique that can help here is called *functional decomposition* [Wes12, p. 78], which consists of breaking down a complex entity into smaller, more manageable parts. Here, the application is decomposed into its constituent processes.

For the example case, materials management is identified as the most relevant process of

the ERP system in the given context.

3. Identify data requirements For each process that has resulted from the previous step, the data requirements need to be identified. These requirements describe which datasets are needed as input to a process and the characteristics they must have, e. g., the data schema or the minimum level of data quality. The data requirements can be expressed as a set of metadata, i. e., a data profile.

4a. Verify fulfillment -or- 4b. Discover suitable data Once the data requirements have been identified, there are two options. First, if the data source is already present within the organization, it can be checked whether the imposed requirements are fulfilled. For this, the data profiling variant *data profile validation* (cf. Section 5.4) can be used. The result of this check allows an assessment of whether the existing data source can be used for the intended process. If that is not the case, or there is no data source to begin with, the second option can be taken, which is to discover suitable data. This step can be performed using the data discovery method as described in Section 5.6. It takes the data requirements as input to a search process that results in datasets that fulfill them.

In the ERP case, the data for the materials management process must contain detailed descriptions about current inventory levels that are both current and accurate. Profiling of the interal data sources showed that these requirements can be fulfilled by the existing systems.

Example Case: MicroCorp As an example for using the application-first approach, the MicroCorp case is presented. It is loosely inspired by a project seminar that took place at the Department of Information Systems at the University of Münster during the winter term 2016/2017[1]. MicroCorp is a company that sells microphones for industrial and scientific customers, i. e., they operate in a business-to-business (B2B) market. They have two branches: the first is located in Hungary and is responsible for research, development, and manufacturing of their products, while the second is the headquarter, which is located in Münster, Germany, and takes care of sales and marketing.

The microphones of MicroCorp are technologically highly advanced assets that are high in value and low in volume. This means that the revenue is earned with a low amount of sales each month. Most of their operations are thus handled manually. For example, incoming customer orders are recorded and tracked using spreadsheet files. This approach worked well so far, with experienced staff that knew what to do and where to look for information. However, as MicroCorp is slowly expanding their business world-wide, a more scalable solution is required, in particular with regards to inventory control, workflow support, and reporting capabilities.

The goal of the project was to analyze the requirements of MicroCorp and recommend an appropriate solution. Such an analysis requires the consideration of the complete application stack, and thus, the GAF was considered to be a good fit.

[1]For compliance with non-disclosure agreements, names and references have been omitted. If needed, the complete report of the project seminar can be obtained from the author of this thesis.

In the first step, the application was profiled and all relevant metadata gathered. It was determined that a CRM system would be a good fit for MicroCorp, as that would not only enable them to track their inventory, but also support many of their current operations. Implementing such a system was not feasible due to the inherent complexity and MicroCorp's lack of IT personnel. Thus, a third-party solution was sought, which could be customized to fit their needs.

Next, the relevant processes were identified. These could be grouped into the classes inventory management, lead management, marketing, sales management, order management, accounting, and human resources. Prior to this project, MicroCorp did not have any explicit process models in place to describe all these processes. Instead, most of the employees knew from experience how things worked. Thus, project participants set out to explicate this implicit knowledge by interviewing key employees and reviewing operation manuals. These efforts resulted in numerous process models, which gave a precise representation of how MicroCorp conducts their business.

The processes were modeled as Petri nets that contained object models to represent the entities that were passed in between individual steps. These object models could be analyzed to show the exact requirements placed upon the data that these processes needed. From these requirements, a database schema could be derived that would be able to serve all processes. This schema contained all relevant entities, including inventory items, orders, reservations, parcels and customers.

Lastly, the existing spreadsheet files were matched against the derived database schema in a multi-source profiling process (cf. Section 5.3). Most of the data could be mapped onto the target schema successfully. Some data was not available, like reservations, because these were handled in an offline fashion by physically placing reserved products into a separate container. In order to fill this gap, the corresponding data would need to be filled in manually.

The final result was a complete picture of every data source and process that needed to be included in the desired CRM system. With this information, a thorough market analysis could be conducted that showed which commercial offerings would be a good fit for the requirements of MicroCorp. The results of this analysis were used to make an educated decision on which CRM system to buy and the project came to a successful conclusion.

7.3.3 Process-First Approach

The third approach applies in cases where the process is the starting point into the GAF. From this middle level, efforts can be made to go upwards and identify enabled applications, or reach into the lower level and identify data requirements. This approach is thus a hybrid between the first two approaches and combines elements from both the top-down and the bottom-up directions, as shown in Figure 7.4.

As the first step, a profiling of the given process is recommended, as described in Section 7.2.1. This provides an overview of what the process does and where it is applied. The remaining steps are re-utilizations of some of the steps that have been described before. Identifying enabled applications was already introduced as part of the data-first approach in Section 7.3.1. The other steps were described in Section 7.3.2 within the application-first approach. In an ideal setting, these two directions (up and down) can be taken in parallel,

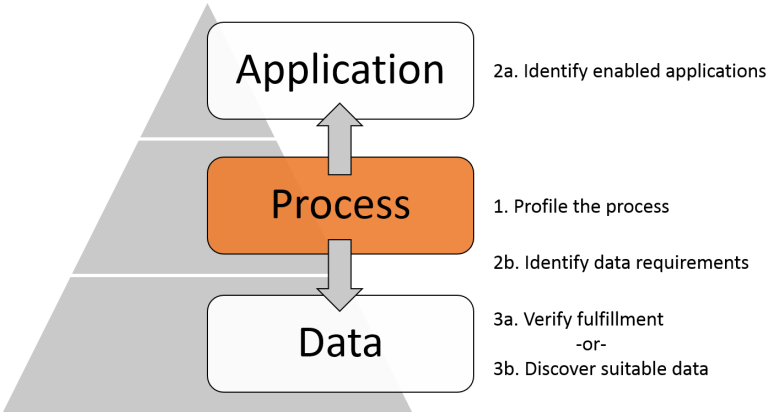

Figure 7.4: Process-first approach.

because there are no direct interdependencies between the data and the application layer. It is admitted however that this may not be true in a practical setting, so it is recommended to perform the steps in sequence and check for interdependencies along the way.

Example Case: MultiChannelSupport To demonstrate how the process-first approach can be applied, the example case of MultiChannelSupport is introduced. Again, this is a fictitious case that is loosely inspired by a real project that started in March 2016 at the Department of Information Systems at the University of Münster. In this case, the subsidiary MultiChannelSupport offers customer support services to its parent company. These services are offered on multiple channels, including social media, telephone, email and live chat.

At the core of MultiChannelSupport's business model is the customer journey shown in Figure 7.5. It shows the different phases a customer goes through, ranging from pre-sales over sales to after-sales. Within these phases, multiple subprocesses are included that go from the selection and acquisition of customers, to the extension of the customer base and the retention of existing customers. Along this journey, a customer may have various questions or concerns that he wishes to discuss with the company. To do so, he can choose one of many different channels to establish contact. The goal of MultiChannelSupport is to record these interaction across all these various channels and integrate them into one unified view of the customer. This enables, for example, a customer service representative during a call about an after-sales issue to immediately look into the history of the customer that is on the line and see all previous interactions, independent of how they occurred.

With the customer journey in place as a focal process, MultiChannelSupport chooses to use the process-first approach of the GAF to reach their goal. At the start, the process is profiled thoroughly. Being a super process, it is necessary to drill down into the respective subprocesses and profile them as well.

Then, the application level is considered. As the customer journey revolves around the customer and his interactions, it makes sense to consider a CRM system. Many of the process metadata can be used to derive the requirements that a suitable CRM system should have. For example, a high number of concurrent users must be supported, because many different customer service representatives need to use the system at the same time. Furthermore, a

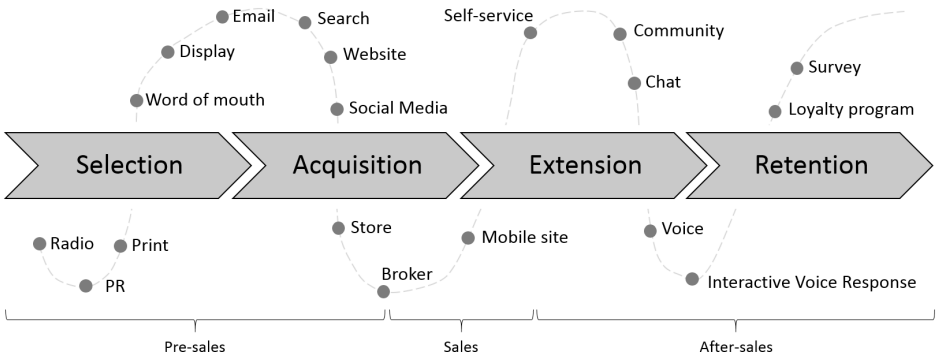

Figure 7.5: Customer journey. Source: [TVH+17].

high availability of the system must be guaranteed, because it is a very crucial part of the business and every outage is likely to lead to customer dissatisfaction and thus lost profit. Therefore, MultiChannelSupport should consider to allocate a high budget for the purchase of a CRM system as basis for their operations to ensure that these requirements can be met.

Next, attention is shifted towards the data level. There is no explicit input data that the customer journey process needs from outside sources. Instead, the required data is created by the process itself, in earlier steps. One of the core issues in the management of this data is that it structured in different formats. For example, the customer sales history is stored in a relational database, while social media interactions are stored as XML documents. This necessitates an integration procedure that can deal with heterogeneous data models. Further considerations must be taken with regards to how the customer data is processed and for how long it is stored, because it represents personally identifiable data. Such data is subject to strict laws and regulations, such as the General Data Protection Regulation in the European Union [39].

In the end, MultiChannelSupport has a complete picture of how the initial process influences the superordinate application and the subordinate level. This information provides them with a solid foundation from which further decisions and plans to reach their business goals can be made.

7.3.4 Combined Approaches

There may exist cases in which components of more than one level may be available at the start, i. e., application and process, application and data, process and data, or all three of these. In these cases, a combination of the described steps can be used. For example, if the application is in place as well as the data, then the user can choose to either approach the process level from above or below. Alternatively, he might even do both simultaneously, and then match the results in the middle. In concrete terms this would lead to a scenario where a given dataset is used to identify enabled processes, and an application is assessed for relevant processes at the same time. Ideally, the results of these two approaches match, and the processes of both are the same. If there is a mismatch, the user has gained valuable knowledge and should reassess his input elements.

8 Conclusion

To conclude this thesis, Section 8.1 gives a summary of each chapter and outlines its contributions. Then, Section 8.2 provides an outlook on open issues and research questions in the context of data profiling that require further work.

8.1 Summary

The state of the art of data profiling The first part of this thesis addressed the task to establish a definition of data profiling and provide an overview of its constituent components. To this end, Chapter 2 started with a presentation of various application areas that use the term *profiling* as part of their description. Seven different areas (cf. Sections 2.2.1 to 2.2.7) have been identified and analyzed with regards to their objective, input, processing and output. Furthermore, they have been classified according to their main purpose, which could range from description over prediction to prescription. The results from these observations have been used to derive a generic model for profiling activities, which was used later on to represent a data profiling process.

Chapter 3 then provided more detail about the problem that data profiling attempts to solve by characterizing its individual parts. First, the input datasets were considered with a specific focus on the data model. The data model of a dataset is of particular importance because it is a deciding factor in the selection of profiling methods, because not every metadata type is applicable under every data model. Next, a list of seven typical tasks (cf. Section 3.2) was described that frequently occur in a data-centric environment and can benefit from data profiling. This was followed by a description of the increasingly common problem of information overload, which can be observed in the preceding tasks. The solution to this problem, and the overall goal state to strive for, is data comprehension, which is achieved when the data in question is understood to a sufficient degree. In particular, data comprehension is dependent on the three aspects (i) dataset, (ii) task, and (iii) user, i. e., a user must understand the dataset that is relevant for a task in order to achieve it. The user itself can be characterized according to his technical expertise and his domain knowledge. These two dimensions were used to establish a classification scheme that highlights the two user classes *domain expert* (high domain knowledge, low technical expertise) and *IT professional* (low domain knowledge, high technical expertise). The individual roles of these classes and their interrelationship are fundamental for the later parts. The chapter closed with an observation about common solution strategies that are employed by people that are unaware of data profiling techniques. These strategies are mostly ad-hoc approaches that tend to fail for various reasons, which is one of the reasons why data profiling should be applied more rigorously.

This led to Chapter 4, in which in-depth descriptions of what data profiling is precisely

were given. To begin, a definition of the term *data profiling* was established in Section 4.1 and consequently compared to and verified against definitions by other authors. They all agreed on the fact that data profiling is concerned with the extraction of metadata. What precisely metadata is, of which types it is comprised, and how they can be put to use were the central questions answered in Section 4.2, where a typology on metadata was presented. It consisted of a discussion of metadata types of the classes *single field*, *multiple fields*, and *multiple datasets*. The typology was completed by a presentation of classification schemes that arrange the individual types into semantic groups. In the next section, the perspective was shifted to look at data profiling in terms of the IPO model, i. e., input, processing, output. This provides an overall framework into which specific considerations of future academic work can be integrated. The last section in this chapter presented an overview of selected software tools and research projects. This provided the reader with a practical insight into how the so-far theoretical considerations can be realized in practice and made usable.

Chapter 5 expanded upon the IPO perspective on profiling by also including the purpose, the executor, the user, and application areas in order to establish a characterization framework. This framework was used to describe six different variants of data profiling that differ in their individual inputs and outputs. Lastly, all these variants were brought together into one unified, high-level data profiling process.

Profiling beyond data The second part of this thesis aimed to expand profiling and its methods to the context of data. This was achieved by first broadening the scope of metadata to also include the extrinsic side in Chapter 6. Extrinsic metadata was defined as metadata that cannot be derived solely from the data itself. These types of metadata are highly useful in numerous scenarios, as was demonstrated in small examples. While some of the shown concepts were not new discoveries, such as data provenance or data validity, their interpretation in a data profiling context was a novel contribution. Furthermore, the sources and means of extraction of extrinsic metadata have been introduced and brought together in a structured process model. The chapter concluded with a combined metadata extraction process and a hierarchical data profile type overview, both of which unify the previously discussed intrinsic view with the newly proposed extrinsic view.

Chapter 7 shifted the perspective towards the context in which data usually resides and introduced the GAF as a means of conceptualization. The GAF consists of the three levels *application*, *process*, and *data*, onto which many application scenarios can be mapped. For every level of the GAF, a list of metadata was introduced, which enabled a profiling on each of them. Furthermore, three different approaches to move between the levels have been proposed that start at each of the levels. The data-first approach can be applied when data is available, and enabled processes and applications are sought in a bottom-up fashion. This is useful, e. g., when an organization has a dataset stored, but is unsure on what it can be used for. Next, the application-first approach corresponds to a top-down perspective in which the application is specified at the beginning. From there, relevant processes and datasets are identified, which are needed for a successful implementation of the application. This resembles a more traditional approach to IT projects that start from the goal and derive requirements that must be met. The third approach is a hybrid that starts in the middle from

the process level, which may not be as often encountered as the other two approaches, but can still be of use when, for example, reference processes are set up first without specifying the applications and data sources. Overall, the assumption is put forth that moving between the levels brings the advantage of extending the scope of consideration, which leads to a more holistic view. Each approach was described in-depth by means of an example that was inspired by real projects. This demonstrated the applicability and usefulness of the proposed approaches.

8.2 Outlook

One of the bigger IT hypes of the recent years has been big data. Referred to as "the big buzzword of 2013" [Vos14, p. 3], it is used as a catchall phrase to describe the development of organizations losing the ability to efficiently and effectively handle the data at their disposal [CML14]. There are multiple reasons for this, which are usually referred to as a set of V's. The original set consisted of the three V's volume, velocity, and variety [Lan01]. Each of these V's represents an area that also affects data profiling.

Volume: Profiling large datasets As data processing algorithms scale with the size of the input data, increasing volumes tend to be a problem for non-efficient and unoptimized algorithms. Thus, naive implementations of data profiling algorithms may quickly reach the point at which their execution is no longer feasible as they simply take too long. For example, the discovery of inclusion dependencies has been shown to be PSPACE-complete [CFP84]. This necessitates tradeoffs to be made that sacrifice precision in favor of more efficient runtimes. There has already been done plenty of work in this regard, e. g. efficient IND discovery is described in [BLNT07] and [PKQRN15], but there is still potential to design better optimized algorithms that handle large volumes either more effectively, or lose less precision in their results.

Velocity: Profiling data streams Every input dataset considered in this thesis was always assumed to be available in a fixed and static format. There are, however, cases in which this assumption does not hold, and the data is received over a continuous time span. For example, a social media data source generates data non-stop as users post statuses and updates, which constitutes a data stream [Agg06]. One way to deal with data streams is called *micro batching* and relies on cutting the stream into small subsets, which are treated as a small static set of data [HYG+10]. Depending on the chosen size of these batches, this approach can be quite memory expensive and may not be always feasible.

In true continuous stream processing, the data comes in at a very high velocity, which means there is little time to perform computations on it. Usually, every data item is only seen once, because memory must be freed for the subsequent data, and there is not enough space available for storage. This means that the algorithm must be executed fast enough to process the data while it is in memory, and any algorithm that requires multiple passes over the data cannot be used.

Several profiling algorithms are affected by this. For example, duplicate detection usually relies on comparing data instances with each other, which is not possible in a stream. As a compromise, an approximation for duplicate detection can be implemented using Stable Bloom Filters. This has been demonstrated by Deng and Rafiei [DR06]. Further work needs to be put into identifying a similar approximation for other profiling algorithms that rely on multiple passes, such as the detection of inclusion or functional dependencies, or the discovery of foreign keys.

Variety: Profiling non-relational data Previous research in the area of data profiling tends to focus on one specific data model, usually the relational one. For example, Olson states that the "target data for [his] book is structured data captured in corporate databases" [Ols03, p. XVI], and Abedjan et al. say that they "focus [their] discussion on relational data, the predominant format of traditional data profiling methods" [AGN15, p. 558].

There exists work that focuses on specific non-relational data models. ProLOD is a tool developed by Böhm et al. that is specifically designed to profile linked open data (LOD) [BNA+10]. Since its inception, ProLOD has been extended and enhanced into ProLOD++ [AGJN14]. As part of the Semantic Web, LOD is data that uses a graph-based structure, such as the Resource Description Framework (RDF) [41], as an interconnection mechanism ("linked") and is publicly available ("open"). The authors rightfully argue that due to the heterogeneity and volume of LOD, classical profiling techniques are not appropriate for dealing with them. They adapt data mining techniques, specifically clustering, to overcome these issues and create an intermediate structure upon which traditional profiling methods can then be executed.

The question remains whether relational profiling techniques need to be specifically adapted for each individual data model, or whether it is possible to develop one unifying solution that bridges the gaps between the individual data models and provides a generalized set of profiling methods that can be applied anywhere, thus saving lots of effort. While the answer to that question is out of the scope of this thesis, an initial idea on how to achieve such a data model-agnostic generalization is presented here.

In 1976, Chen wrote that "the entity-relationship model can be used as a basis for unification of different views of data" [Che76, p. 9]. Entity-relationship modeling (ERM) is a technique that is used to model the entities within a domain and how they relate to each other. An ER model is a *conceptual* model, which means that in order to actually store data, it needs to be transformed into a physical model. This transformation process is well-described for a relational model as the target [Vos91, p. 104]. However, following Chen's words, it is also possible to transform an ER model into any other data model, such as a document-oriented or graph-oriented one. This fact is visualized in Figure 8.1, where arrows represent a transformation process.

It is also possible to reverse the arrows and go from an implemented physical model back to the conceptual level. This process is called *reverse engineering* and was proven to be possible for the relational case [Fah96, p. 151]. If this proof can be extended to also include the other data models, it would be possible to move horizontally from one physical model to another, using the ER level as an intermediate step.

Using the same logic, it should be possible to take relational profiling methods, raise them to the conceptual level, and then bring them down to another target physical data model.

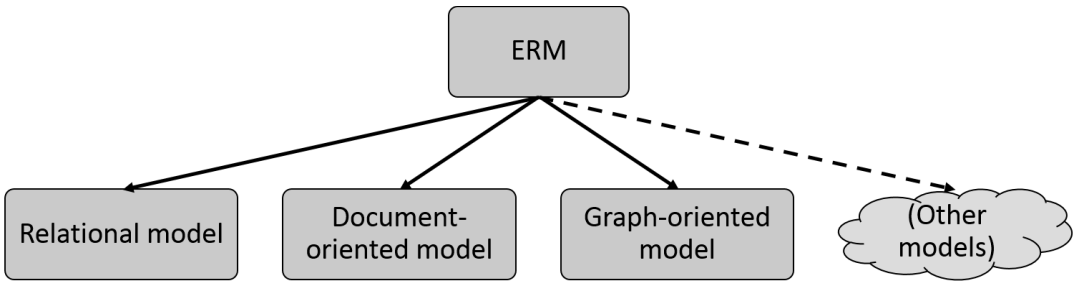

Figure 8.1: Relation between ERM and data models.

However, there are some issues that would need to be overcome before this idea can become reality. First, a formalization of profiling methods is required that allows their representation on a conceptual model. Second, it must be guaranteed that all transformations are correctly invertible, i. e., executing a transformation and then reversing it leads back to the exact same input.

Meaningful interpretation of data profiles Just like data, metadata needs to be interpreted by users to gain value. Any data profiling effort is worthless if the resulting metadata cannot be interpreted correctly. It has already been pointed out in Section 4.3.3 that this is the hardest part of data profiling [AGN15, p. 558], because it cannot be automated or standardized efficiently. This open issue is not satisfyingly solved so far. One solution that could be attempted is to establish a guide that consists of best practices for interpreting metadata. This guide could be based on the list of metadata types put forward in this thesis and expand upon it by providing typical values and what insights can be derived from them. For example, if an inclusion dependency is detected across tables from different data sources, then it is likely that these can be joined together. There are already examples of what can be done with some metadata types, but there is an opportunity to expand these into a thorough data profile interpretation guide.

Bibliography

[ABD+97] Jennifer M Anderson, Lance M Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R Henzinger, Shun-Tak A Leung, Richard L Sites, Mark T Vandevoorde, Carl A Waldspurger, and William E Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, pages 1–14, New York, NY, USA, 1997. ACM.

[ABU79] Alfred Aho, Catriel Beeri, and Jeffrey Ullman. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.*, 4(3):297–314, sep 1979.

[Ack89] Russell L Ackoff. From Data to Wisdom. *Journal of Applied Systems Analysis*, 16(1):3–9, 1989.

[AG08] Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Computing Surveys*, 40(1):1–39, feb 2008.

[Agg06] Charu C Aggarwal. *Data Streams: Models and Algorithms*. Advances in Database Systems. Springer, 2006.

[Agg15] Charu C Aggarwal. *Data Mining: The Textbook*. Springer, 2015.

[AGJN14] Ziawasch Abedjan, Toni Gruetze, Anja Jentzsch, and Felix Naumann. Profiling and Mining RDF Data with ProLOD++. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1198–1201. IEEE, mar 2014.

[AGN15] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling Relational Data: A Survey. *VLDB Journal*, 24(4):557–581, 2015.

[AIS+93] Rakesh Agrawal, Tomasz Imieliński, Arun Swami, Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, volume 22 (2), pages 207–216, New York, New York, USA, 1993. ACM Press.

[AL01] Maryam Alavi and Dorothy E. Leidner. Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. *MIS Quarterly*, 25(1):107, mar 2001.

[AN95] Agnar Aamodt and Mads Nygård. Different Roles and Mutual Dependencies of Data, Information, and Knowledge — An AI Perspective on their Integration. *Data & Knowledge Engineering*, 16(3):191–222, sep 1995.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[AZ12] Charu C. Aggarwal and ChengXiang. Zhai. *Mining Text Data*. Springer, 2012.

[BAL+12] Jana Bauckmann, Ziawasch Abedjan, Ulf Leser, Heiko Müller, and Felix Naumann. Discovering Conditional Inclusion Dependencies. In *Proceedings of the 21st ACM international conference on Information and knowledge management - CIKM '12*, page 2094, New York, New York, USA, 2012. ACM Press.

[BB17] Paul Begg and John Bennett. *Jack the Ripper*. Andre Deutsch Ltd, 2017.

[BCFM09] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for Data Quality Assessment and Improvement. *ACM Computing Surveys*, 41(3):1–52, jul 2009.

[Ben93] David Benyon. Accommodating Individual Differences through an Adaptive User Interface. *Adaptive User Interfaces - Results and Prospects*, 10:1–16, 1993.

[BK01] S. Bassil and R. K. Keller. Software Visualization Tools: Survey and Analysis. In *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, pages 7–17. IEEE Comput. Soc, 2001.

[BK14] Dimitris Bertsimas and Nathan Kallus. From Predictive to Prescriptive Analytics. Technical report, Massachusetts Institute of Technology, feb 2014.

[BKR14] Joerg Becker, Martin Kugeler, and Michael Rosemann. *Process Management: A Guide for the Design of Business Processes*. Springer, 2014.

[BKWC01] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and Where: A Characterization of Data Provenance. In Jan den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 316–330, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BL94] Thomas Ball and James R. Larus. Optimally Profiling and Tracing Programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1319–1360, jul 1994.

[BLNT07] Jana Bauckmann, Ulf Leser, Felix Naumann, and Veronique Tietz. Efficiently Detecting Inclusion Dependencies. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1448–1450. IEEE, 2007.

[BM15] Francis Buttle and Stan Maklan. *Customer Relationship Management: Concepts and Technologies*. Routledge, 3rd edition, 2015.

[BMU+97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, Shalom Tsur, Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *ACM SIGMOD Record*, 26(2):255–264, jun 1997.

[BNA+10] Christoph Böhm, Felix Naumann, Ziawasch Abedjan, Dandy Fenz, Toni Grutze, Daniel Hefenbrock, Matthias Pohl, and David Sonnabend. Profiling Linked Open Data with ProLOD. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pages 175–178. IEEE, mar 2010.

[Bou04] Mokrane Bouzeghoub. A Framework for Analysis of Data Freshness. In *Proceedings of the 2004 international workshop on Information quality in informational systems - IQIS '04*, page 59, New York, New York, USA, 2004. ACM Press.

[BS06] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer Berlin Heidelberg, 2006.

[BTN90] Dermont Browne, Peter Totterdell, and Mike Norman. *Adaptive User Interfaces (Computers and People Series)*. Academic Press Inc, 1990.

[BV05] Aida Boukottaya and Christine Vanoirbeek. Schema Matching for Transforming Structured Documents. In *Proceedings of the 2005 ACM symposium on Document engineering - DocEng '05*, page 101, New York, New York, USA, 2005. ACM Press.

[Can94] David V. Canter. *Criminal shadows: Inside the Mind of the Serial Killer*. Harper Collins, 1994.

[CC90] Elliot J. Chikofsky and J. H. Cross. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.

[CCM17] Adriane Chapman, James Cheney, and Simon Miles. Guest Editorial: The Provenance of Online Data. *ACM Transactions on Internet Technology*, 17(4):1–3, aug 2017.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, mar 1997.

[CDDF16] Fernando Chirigati, Harish Doraiswamy, Theodoros Damoulas, and Juliana Freire. Data Polygamy. In *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*, pages 1011–1025, New York, New York, USA, 2016. ACM Press.

[CFH05] Adrienne Curry, Peter Flett, and Ivan Hollingsworth. *Managing Information and Systems: The Business Perspective*. Routledge, 2005.

[CFP84] Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, feb 1984.

[Cha98] Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '98*, pages 34–43, New York, New York, USA, 1998. ACM Press.

[Che76] Peter Pin-Shan Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, mar 1976.

[Chr12] Peter Christen. *Data Matching: Concepts and Techniques for Record linkage, Entity Resolution, and Duplicate Detection.* Springer Berlin Heidelberg, 2012.

[CIPY14] Xu Chu, Ihab F. Ilyas, Paolo Papotti, and Yin Ye. RuleMiner: Data Quality Rules Discovery. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1222–1225. IEEE, mar 2014.

[CK07] Varun Chandola and Vipin Kumar. Summarization – Compressing Data into an Informative Representation. *Knowledge and Information Systems*, 12(3):355–378, aug 2007.

[CLD+12] Francesco Contino, Tommaso Lucchini, Gianluca D'Errico, Catherine Duynslaegher, Veronique Dias, and Herve Jeanmart. Simulations of Advanced Combustion Modes Using Detailed Chemistry Combined with Tabulation and Mechanism Reduction Techniques. *SAE International Journal of Engines*, 5(2):2012–01–0145, apr 2012.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, apr 2014.

[CMS10] W. Bruce. Croft, Donald. Metzler, and Trevor. Strohman. *Search Engines: Information Retrieval in Practice.* Addison-Wesley, 2010.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, jun 1970.

[Dav12] E. R. Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities.* Elsevier, 2012.

[DB00] Evelyn Duesterwald and Vasanth Bala. Software Profiling for Hot Path Prediction. *ACM SIGPLAN Notices*, 35(11):202–211, nov 2000.

[DFAB03] Alan Dix, Janet E Finlay, Gregory D Abowd, and Russell Beale. *Human-Computer Interaction (3rd Edition).* Pearson, 2003.

[DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. Principles of Data Integration. *Principles of Data Integration*, pages 95–119, 2012.

[Die17] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics).* Springer, 2017.

[DJM06] Tamraparni Dasu, Theodore Johnson, and Amit Marathe. Database Exploration Using Database Dynamics. *IEEE Data Eng. Bull.*, 29(2):43–59, 2006.

[DJMS02] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining Database Structure; or, How to Build a Data Quality Browser. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data - SIGMOD '02*, page 240, New York, New York, USA, 2002. ACM Press.

[DLP09] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. Unary and n-ary Inclusion Dependency Discovery in Relational Databases. *Journal of Intelligent Information Systems*, 32(1):53–73, feb 2009.

[dOL03] M.C.F. de Oliveira and H. Levkowitz. From Visual Data Exploration to Visual Data Mining: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, jul 2003.

[DR06] Fan Deng and Davood Rafiei. Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, page 25, New York, New York, USA, 2006. ACM Press.

[DRBH86] John E. Douglas, Robert K. Ressler, Ann W. Burgess, and Carol R. Hartman. Criminal Profiling from Crime Scene Analysis. *Behavioral Sciences & the Law*, 4(4):401–421, 1986.

[DV12] Martin Dugas and Gottfried Vossen. CityPlot: Colored ER Diagrams to Visualize Structure and Contents of Databases. *Datenbank-Spektrum*, 12(3):215–218, nov 2012.

[Dwo08] Cynthia Dwork. Differential Privacy: A Survey of Results. In *Theory and Applications of Models of Computation*, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[EC05] Eldad Eilam and Elliot J Chikofsky. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005.

[EM00] Angela Edmunds and Anne Morris. The Problem of Information Overload in Business Organisations: A Review of the Literature. *International Journal of Information Management*, 20(1):17–28, 2000.

[EN15] Ramez Elmasri and Sham Navathe. *Fundamentals of Database Systems*. Pearson, 7th edition, 2015.

[ESJ10] Mturi Elias, Khurram Shahzad, and Paul Johannesson. A Business Process Metadata Model for a Process Model Repository. *Lecture Notes in Business Information Processing*, pages 287–300, 2010.

[Fag81] Ronald Fagin. A Normal Form for Relational Databases that is Based on Domains and Keys. *ACM Transactions on Database Systems*, 6(3):387–415, sep 1981.

[Fah96] Christian Fahrner. *Schematransformationen in Datenbanken.* Phd thesis, WWU Muenster, 1996.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* Dissertation, University of California, Irvine, 2000.

[FMR12] Catherine O. Fritz, Peter E. Morris, and Jennifer J. Richler. Effect Size Estimates: Current Use, Calculations, and Interpretation. *Journal of Experimental Psychology: General*, 141(1):2–18, 2012.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge Discovery and Data Mining: Towards a Unifying Framework, 1996.

[Fri06] Jeffrey E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, 2006.

[FS69] Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, dec 1969.

[GIJ+01] Luis Gravano, Panagiotis G Ipeirotis, H V Jagadish, Nick Koudas, S Muthukrishnan, and Divesh Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 491–500, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[GKKS10] Lukasz Golab, Howard Karloff, Flip Korn, and Divesh Srivastava. Data Auditor: Exploring Data Quality and Semantics Using Pattern Tableaux. *Proceedings of the VLDB Endowment*, 3(1-2):1641–1644, 2010.

[GL12] Jan Goyvaerts and Steven Levithan. *Regular Expressions Cookbook.* O'Reilly and Associates, 2012.

[Goe10] Anita Goel. *Computer Fundamentals.* Pearson Education India, 2010.

[GR93] Jim Gray and A. (Andreas) Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, 1993.

[Gra10] Jeffrey O. Grady. *System Management: Planning, Enterprise Identity, and Deployment.* CRC Press, 2010.

[Gra12] Dan Graham. The Data Temperature Spectrum (Teradata white paper), 2012.

[Hal01] Alon Y. Halevy. Answering Queries Using Views: A Survey. *The VLDB Journal*, 10(4):270–294, dec 2001.

[HCC11] D. Ian. Heywood, Sarah. Cornelius, and Steve. Carver. *An Introduction to Geographical Information Systems.* Prentice Hall, 2011.

[HGN00] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for Association Rule Mining — A General Survey and Comparison. *ACM SIGKDD Explorations Newsletter*, 2(1):58–64, jun 2000.

[HHK+01] Joseph Hall, Jason Hartline, Anna R Karlin, Jared Saia, and John Wilkes. On Algorithms for Efficient Data Migration. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 620–629, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[HLK+12] Joseph M. Hellerstein, Kun Li, Arun Kumar, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, and Xixuan Feng. The MADlib Analytics Library. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, aug 2012.

[HMM00] I. Herman, G. Melancon, and M.S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[HN17] Hazar Harmouch and Felix Naumann. Cardinality Estimation: An Experimental Survey. *PVLDB*, 11(4):499–512, 2017.

[HQRA+13] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. Scalable Discovery of Unique Column Combinations. *Proceedings of the VLDB Endowment*, 7(4):301–312, dec 2013.

[HT85] Starr R. Hiltz and Murray Turoff. Structuring Computer-Mediated Communication Systems to Avoid Information Overload. *Communications of the ACM*, 28(7):680–689, jul 1985.

[HT03] Anthony J G Hey and Anne E Trefethen. The Data Deluge: An e-Science Perspective. In F. Berman, G. C. Fox, and A. J. G. Hey, editors, *Grid Computing - Making the Global Infrastructure a Reality*, pages 809–824. Wiley and Sons, 2003.

[HVL08] Niels Haering, Péter L. Venetianer, and Alan Lipton. The Evolution of Video Surveillance: An Overview. *Machine Vision and Applications*, 19(5-6):279–290, oct 2008.

[HXHLB14] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm Runtime Prediction: Methods and Evaluation. *Artificial Intelligence*, 206:79–111, jan 2014.

[HYG+10] Bingsheng He, Mao Yang, Zhenyu Guo, Rishan Chen, Bing Su, Wei Lin, and Lidong Zhou. Comet: Batched Stream Processing for Data Intensive Distributed Computing. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, page 63, New York, New York, USA, 2010. ACM Press.

[IJ90] J. Iivari and J. Implementation of In-House Developed vs Application Package Based Information Systems. *ACM SIGMIS Database*, 21(1):1–10, sep 1990.

[Inm05] W H Inmon. *Building the Data Warehouse*. Wiley, 2005.

[IOF08] William H. Inmon, Bonnie K. O'Neil, and Lowell. Fryman. *Business Metadata: Capturing Enterprise Knowledge*. Elsevier/Morgan Kaufmann, 2008.

[ISO04] ISO. ISO/IEC 11179-1:2004(E): Information Technology - Metadata Registries, 2004.

[JCK03] Jiming Liu, Chi Kuen Wong, and Ka Keung Hui. An Adaptive User Interface Based on Personalized Learning. *IEEE Intelligent Systems*, 18(2):52–57, mar 2003.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, 2013.

[JWT85] Alec J. Jeffreys, Victoria Wilson, and Swee Lay Thein. Hypervariable 'Minisatellite' Regions in Human DNA. *Nature*, 314(6006):67–73, mar 1985.

[KA01] Daniel A. Keim and Daniel A. Visual Exploration of Large Data Sets. *Communications of the ACM*, 44(8):38–44, aug 2001.

[KAF+08] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual Analytics: Definition, Process, and Challenges. In *Information Visualization*, pages 154–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Kei02] D.A. Keim. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[KH13] Axel. Kuhn and Bernd. Hellingrath. *Supply Chain Management: Optimierte Zusammenarbeit in der Wertschoepfungskette*. Springer, 2013.

[Kle51] Stephen Cole Kleene. Representation of Events in Nerve Nets and Finite Automata (Research memo), 1951.

[KPP+12] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces - AVI '12*, page 547, New York, New York, USA, 2012. ACM Press.

[KR13] Ralph. Kimball and Margy. Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley, 2013.

[KRB+16] Ralph Kimball, Margy Ross, Bob Becker, Joy Mundy, Warren Thornthwaite, and Kimball Group. *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence*. John Wiley & Sons, 2016.

[KWdRM03] Wagner A. Kamakura, Michel Wedel, Fernando de Rosa, and Jose Afonso Mazzon. Cross-Selling Through Database Marketing: A Mixed Data Factor Analyzer for Data Augmentation and Prediction. *International Journal of Research in Marketing*, 20(1):45–65, mar 2003.

[Lan01] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety (research note). Technical report, META Group, 2001.

[LAW02] Chenyang Lu, Ga Alvarez, and John Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. *FAST'02 Proceedings of the 1st USENIX conference on File and storage technologies*, 1(January):219–230, 2002.

[LCWL14] Shixia Liu, Weiwei Cui, Yingcai Wu, and Mengchen Liu. A Survey on Information Visualization: Recent Advances and Challenges. *The Visual Computer*, 30(12):1373–1393, dec 2014.

[LL95] Kenneth C. Laudon and Jane P. Laudon. *Essentials of Management Information Systems*. Prentice Hall Inc., 1995.

[LL17] Kenneth C. Laudon and Jane P. Laudon. *Management Information Systems: Managing the Digital Firm*. Pearson Higher Education, 15th edition, 2017.

[LLS10] K. Laudon, J. Laudon, and D. Schoder. *Wirtschaftsinformatik: Eine Einführung*. Pearson Deutschland, 2010.

[LM02] Maurizio Lenzerini and Maurizio. Data Integration. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '02*, page 233, New York, New York, USA, 2002. ACM Press.

[LO96] David Lewis and Others. Dying for Information. *Reuters Business Information, London*, 1996.

[LPT02] Stéphane Lopes, Jean-Marc Petit, and Farouk Toumani. Discovering Interesting Inclusion Dependencies: Application to Logical Database Tuning. *Information Systems*, 27(1):1–19, mar 2002.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014.

[LS87] Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, jan 1987.

[Mar05] Richard Marsh. Drowning in Dirty Data? It's Time to Sink or Swim: A Four-Stage Methodology for Total Data Quality Management. *Journal of Database Marketing & Customer Strategy Management*, 12(2):105–112, jan 2005.

[Mas11] Mark Masse. *REST API Design Rulebook*. O'Reilly, 2011.

[May07] Arkady Maydanchik. *Data Quality Assessment*. Technics Publications, 2007.

[MCS88] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical Profile Estimation in Database Systems. *ACM Computing Surveys*, 20(3):191–221, sep 1988.

[MG13] E. B. Mandinach and E. S. Gummer. A Systemic View of Implementing Data Literacy in Educator Preparation. *Educational Researcher*, 42(1):30–37, jan 2013.

[MPdS+08] Fabio Perez Marzullo, Rodrigo Novo Porto, Geraldo Zimbrao da Silva, Jano Moreira de Souza, and Jose Roberto Blaschek. An MDA Approach for Database Profiling and Performance Assessment. In Roger Lee, editor, *Computer and Information Science*, pages 1–10. Springer Berlin Heidelberg, 2008.

[MWA+07] Zhilei Ma, Branimir Wetzstein, Darko Anicic, Stijn Heymans, and Frank Leymann. Semantic Business Process Repository. *Proceedings of SBPM*, pages 92–100, 2007.

[Mya06] Glenn J Myatt. *Making Sense of Data: A Practical Guide to Exploratory Data Analysis and Data Mining*. Wiley-Interscience, 2006.

[Nau13] Felix Naumann. Data Profiling Revisited. *ACM SIGMOD Record*, 42(4):40–49, feb 2013.

[NEM09] Robert. Nisbet, John F. (John Fletcher) Elder, and Gary. Miner. *Handbook of Statistical Analysis and Data Mining Applications*. Academic Press/Elsevier, 2009.

[Nie05] Detlef Nielsen, editor. *Maritime Security and MET*. WIT Press / Computational Mechanics, 2005.

[NM03] Adithya Nagarajan and Atif M Memon. Refactoring Using Event-Based Profiling. In *Proceedings of The First International Workshop on REFactoring: Achievements, Challenges, Effects*, nov 2003.

[OHB12] Paul Oldham, Stephen Hall, and Geoff Burton. Synthetic Biology: Mapping the Scientific Landscape. *PLoS ONE*, 7(4):e34368, apr 2012.

[O'L00] Daniel Edmund O'Leary. *Enterprise Resource Planning Systems: Systems, Life Cycle, Electronic Commerce, and Risk*. Cambridge University Press, 2000.

[Ols03] Jack E. Olson. *Data Quality—The Accuracy Dimension*. Morgan Kaufmann, 2003.

[Opp97] C. Oppenheim. Managers' Use and Handling of Information. *International Journal of Information Management*, 17(4):239–248, aug 1997.

[PBF+15] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data Profiling with Metanome. *Proceedings of the VLDB Endowment*, 8(12):1860–1863, aug 2015.

[Pea84] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing. Co, 1984.

[PEM+15] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional Dependency Discovery. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, jun 2015.

[Pfl17] Nicolas Pflanzl. *Gamification for Business Process Modeling*. Doctoral thesis, Westfälische Wilhelms-Universität Münster, 2017.

[PKQRN15] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. Divide & Conquer-Based Inclusion Dependency Discovery. *Proceedings of the VLDB Endowment*, 8(7):774–785, feb 2015.

[PLW02] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. Data Quality Assessment. *Communications of the ACM*, 45(4):211, apr 2002.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[Por85] Michael E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press, 1985.

[PSF04] A. Phippen, L. Sheppard, and S. Furnell. A Practical Evaluation of Web Analytics. *Internet Research*, 14(4):284–293, sep 2004.

[Pyl99] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, 1999.

[Ram09] Krish Ramachandran. Adaptive User Interfaces for Health Care Applications. Technical report, IBM, 2009.

[RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, dec 2001.

[RD88] John F. (John Fralick) Rockart and David W. DeLong. *Executive Support Systems: The Emergence of Top Management Computer Use*. Dow Jones-Irwin, 1988.

[RD00] Erhard Rahm and Hong Hai Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[Rei78] Raymond Reiter. On Closed World Data Bases. In *Logic and Data Bases*, pages 55–76. Springer US, Boston, MA, 1978.

[RH01] Vijayshankar Raman and Joseph M Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. *Proceedings of the 27th International Conference on Very Large Data Bases*, 01:381–390, 2001.

[Ril17] Jenn Riley. *Understanding Metadata: What is Metadata, and what is it for?* NISO Press, 2017.

[Rom99] Jorge Luis Romeu. Data Quality and Pedigree (whitepaper). Technical report, IIT Research Institute, Rome, New York, 1999.

[RV97] Paul Resnick and Hal R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, mar 1997.

[SC97] A.J. Stanley and P.S. Clipsham. Information Overload - Myth or Reality? In *IEE Colloquium on IT Strategies for Information Overload*, volume 1997, pages 1–1. IEE, 1997.

[She98] David Shenk. *Data Smog: Surviving the Information Glut.* HarperOne, 1998.

[SHF11] Mauricio Sadinle, Rob Hall, and Stephen E Fienberg. Approaches to Multiple Record Linkage. *Proceedings of International Statistical Institute*, 2011.

[Shn96] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343. IEEE Comput. Soc. Press, 1996.

[Sie16] Eric Siegel. *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die.* Wiley, Hoboken, New Jersey, 2016.

[SKKR00] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of Recommendation Algorithms for e-Commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce - EC '00*, pages 158–167, New York, New York, USA, 2000. ACM Press.

[SMM+09] Kenneth P Smith, Michael Morse, Peter Mork, Maya Hao Li, Arnon Rosenthal, M David Allen, and Len Seligman. The Role of Schema Matching in Large Enterprises. In *{CIDR} 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*. www.cidrdb.org, 2009.

[Sno86] Richard Thomas Snodgrass. Temporal databases. *IEEE COMPUTER*, 19:35—-42, 1986.

[SS13] Nitin Sawant and Himanshu Shah. Big Data Ingestion and Streaming Patterns. In *Big Data Application Architecture Q & A*, pages 29–42. Apress, Berkeley, CA, 2013.

[SSTW01] Michael J Shaw, Chandrasekar Subramaniam, Gek Woo Tan, and Michael E Welge. Knowledge Management and Data Mining for Marketing. *Decision Support Systems*, 31(1):127–137, 2001.

[SSV12] Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for Data: An Initial Survey. *ACM SIGMOD Record*, 42(1):15, may 2012.

[Sta73] Herbert. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Vienna, Austria, 1973.

[Sto16] David Stodder. Improving Data Preparation for Business Analytics (Best Practices Report). Technical report, TDWI, 2016.

[SWC+02] J.P. Shim, Merrill Warkentin, James F. Courtney, Daniel J. Power, Ramesh Sharda, and Christer Carlsson. Past, Present, and Future of Decision Support Technology. *Decision Support Systems*, 33(2):111–126, jun 2002.

[Tal11] John R. Talburt. *Entity Resolution and Information Quality*. Morgan Kaufmann, 2011.

[Tho04] Bruce Thompson. *Exploratory and Confirmatory Factor Analysis: Understanding Concepts and Applications*. American Psychological Association, Washington, 2004.

[TS92] Markus Tresch and Marc H Scholl. Meta Object Management and its Application to Database Evolution. In G Pernul and A M Tjoa, editors, *Entity-Relationship Approach — ER '92*, pages 299–321, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[TSRC15] Ignacio Terrizzano, Peter Schwarz, Mary Roth, and John E. Colino. Data Wrangling: The Challenging Yourney from the Wild to the Lake. In *7th Biennial Conference on Innovative Data Systems Research (CIDR '15)*, Asilomar, California, USA, 2015.

[Tuk77] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley Pub. Co, 1977.

[Tur11] Brent E. Turvey. *Criminal Profiling: An Introduction to Behavioral Evidence Analysis*. Academic Press, 2011.

[TVH+17] Heike Trautmann, Gottfried Vossen, Leschek Homann, Matthias Carnein, and Karsten Kraume. Challenges of Data Management and Analytics in Omni-Channel CRM. Technical report, ERCIS Working Papers, No. 28, 2017.

[VBM95] Marshall Van Alstyne, Erik Brynjolfsson, and Stuart Madnick. Why Not One Big Database? Principles for Data Ownership. *Decision Support Systems*, 15(4):267–284, dec 1995.

[vdA11] Wil M. P. van der Aalst. *Process Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[vdAWM04] W. van der Aalst, T. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, sep 2004.

[VKH15] Jose M Villaveces, Prasanna Koti, and Bianca H Habermann. Tools for Visualization and Analysis of Molecular Networks, Pathways, and -omics Data. *Advances and applications in bioinformatics and chemistry : AABC*, 8:11–22, 2015.

[Vos91] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley Publishing. Co, 1991.

[Vos14] Gottfried Vossen. Big Data as the New Enabler in Business and Other Intelligence. *Vietnam Journal of Computer Science*, 1(1):3–14, feb 2014.

[vZR17] Bjørn Marius von Zernichow and Dumitru Roman. Usability of Visual Data Profiling in Data Cleaning and Transformation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10574 LNCS, pages 480–496. SINTEF, 2017.

[WBW02] K-P Wiedmann, H Buxel, and G Walsh. Customer Profiling in e-Commerce: Methodological Aspects and Challenges. *Journal of Database Marketing & Customer Strategy Management*, 9(2):170–184, jan 2002.

[Wes12] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012.

[WS96] Richard Y. Wang and Diane M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4):5–33, mar 1996.

[WS97] A. Woodruff and M. Stonebraker. Supporting Fine-Grained Data Lineage in a Database Visualization Environment. In *Proceedings 13th International Conference on Data Engineering*, pages 91–102. IEEE Comput. Soc. Press, 1997.

[WT02] Ronald Weitzer and Steven A. Tuch. Perceptions of Racial Profiling: Race, Class, and Personal Experience. *Criminology*, 40, 2002.

[WT07] Jessica Woodhams and Kirsty Toye. An Empirical Test of the Assumptions of Case Linkage and Offender Profiling with Serial Commercial Robberies. *Psychology, Public Policy, and Law*, 13(1):59, 2007.

[Wu03] Youfeng Wu. Software Profiling Method and Apparatus (US Patent 6,668,372). Patent, Intel Corp, 2003.

[YaKS07] Ji Soo Yi, Youn ah Kang, and John Stasko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.

List of Web Pages

[1] Abadi, Daniel: *DBMS Musings: Machine vs. human generated data*. http://dbmsmusings.blogspot.de/2010/12/machine-vs-human-generated-data.html. – Last accessed: 2018-04-17

[2] Bort, Julie: *Amazon CEO Jeff Bezos explains his famous one-character emails , known to strike fear in manager's hearts*. https://finance.yahoo.com/news/amazon-ceo-jeff-bezos-explains-212315127.html. – Last accessed: 2018-04-23

[3] Chamorro-Premuzic, Tomas: *How the web distorts reality and impairs our judgement skills*. https://www.theguardian.com/media-network/media-network-blog/2014/may/13/internet-confirmation-bias. – Last accessed: 2017-06-09

[4] Datamartist: *Datamartist FAQ*. http://www.datamartist.com/product/datamartist-faq. – Last accessed: 2017-07-07

[5] Dictionary.com: *Define Profile at Dictionary.com*. http://www.dictionary.com/browse/profile. – Last accessed: 2017-05-03

[6] Donald, Heather M.: *The Myth of Racial Profiling*. https://www.city-journal.org/html/myth-racial-profiling-12022.html. – Last accessed: 2017-05-18

[7] Dormando: *memcached - a distributed memory object caching system*. https://memcached.org/. – Last accessed: 2018-04-23

[8] Faception: *Facial Personality Profiling*. http://www.faception.com. – Last accessed: 2017-05-12

[9] Friend, Zach: *Predictive Policing: Using Technology to Reduce Crime*. https://leb.fbi.gov/2013/april/predictive-policing-using-technology-to-reduce-crime. – Last accessed: 2017-06-21

[10] GovData: *Datenportal für Deutschland*. https://www.govdata.de/. – Last accessed: 2018-05-06

[11] Harper, Douglas: *Online Etymology Dictionary*. http://www.etymonline.com/index.php?term=profile. – Last accessed: 2017-04-27

[12] IBM: *IBM Knowledge Center - Application profiling*. https://www.ibm.com/support/knowledgecenter/en/SSAW57_7.0.0/com.ibm.websphere.nd.doc/info/ae/appprofile/concepts/capp_overview.html. – Last accessed: 2018-05-03

[13] iSixSigma: *Subject Matter Expert - SME*. https://www.isixsigma.com/dictionary/subject-matter-expert-sme/. – Last accessed: 2018-04-23

[14] James R. Evans, Carl H. L.: *Business Analytics: The Next Frontier for Decision Sciences*. http://www.cbpp.uaa.alaska.edu/afef/business_analytics.htm. – Last accessed: 2017-06-22

[15] LaChapelle, Cindy: *The Cost of Data Storage and Management*. http://www.datacenterjournal.com/cost-data-storage-management-headed-2016/. – Last accessed: 2018-05-08

[16] Lichtblau, Eric: *Bush Issues Federal Ban on Racial Profiling*. http://www.nytimes.com/2003/06/17/politics/bush-issues-federal-ban-on-racial-profiling.html. – Last accessed: 2017-05-23

[17] McKie, Robin: *Eureka moment that led to the discovery of DNA fingerprinting*. https://www.theguardian.com/science/2009/may/24/dna-fingerprinting-alec-jeffreys. – Last accessed: 2017-05-30

[18] Merriam-Webster: *Definition of Metadata*. https://www.merriam-webster.com/dictionary/metadata. – Last accessed: 2018-01-15

[19] Merriam-Webster: *Definition of Profiling*. https://www.merriam-webster.com/dictionary/profiling. – Last accessed: 2017-04-27

[20] Monash, Curt: *Examples and definition of machine-generated data | DBMS2*. http://www.dbms2.com/2010/12/30/examples-and-definition-of-machine-generated-data/. – Last accessed: 2018-04-17

[21] Naumann, Felix: *Metanome Tool and Profiling Algorithms*. https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling/algorithms.html. – Last accessed: 2018-04-14

[22] Oracle: *MySQL 5.7 Reference Manual :: 6.2.1 Privileges Provided by MySQL*. https://dev.mysql.com/doc/refman/5.7/en/privileges-provided.html. – Last accessed: 2018-04-18

[23] Oracle: *Oracle E-Business Suite Developer's Guide*. https://docs.oracle.com/cd/E18727_01/doc.121/e12897/T302934T458266.htm. – Last accessed: 2018-04-23

[24] Palmer, Michael: *ANA Marketing Maestros: Data is the New Oil*. http://ana.blogs.com/maestros/2006/11/data_is_the_new.html. – Last accessed: 2018-05-07

[25] Papenbrock, Thorsten: *Source code for several Metanome data profiling algorithms*. https://github.com/HPI-Information-Systems/metanome-algorithms. – Last accessed: 2018-04-14

[26] Paul Rogers, James R. Rudy Puryear P. Rudy Puryear: *Infobesity: The enemy of good decisions*. http://www.bain.com/publications/articles/infobesity-the-enemy-of-good-decisions.aspx. – Last accessed: 2017-06-09

[27] Prado, Guia Marie D.: *Artificially intelligent security cameras are spotting crimes before they happen*. http://www.businessinsider.com/security-cameras-use-artificial-intelligence-to-detect-crime-2015-8?IR=T. – Last accessed: 2017-05-12

[28] PredPol: *Predictive Policing Software*. http://www.predpol.com/. – Last accessed: 2017-06-21

[29] ProPublica: *Trump's 10 Troubling Deals with Foreign Power-Players*. https://projects.propublica.org/trump-conflicts/. – Last accessed: 2018-05-08

[30] QNX: *Help - Eclipse SDK*. http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.ide.userguide%2Ftopic%2Fprofiler_PROUSECASES_.html. – Last accessed: 2018-05-03

[31] RedisLabs: *Redis*. https://redis.io/. – Last accessed: 2018-04-23

[32] Resnick, Peter: *RFC 5322 - Internet Message Format*. https://tools.ietf.org/html/rfc5322. – Last accessed: 2018-04-24

[33] Schlueter-Isenbeck: *Münster-Skyline*. http://scheidingen.de/muenster-skyline/. – Last accessed: 2018-05-08

[34] Talend: *Open Source Data Quality: Talend Open Studio Data Quality*. https://www.talend.com/products/talend-open-studio/data-quality-open-studio/. – Last accessed: 2018-04-24

[35] Team, Editorial: *The Exponential Growth of Data*. https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/. – Last accessed: 2018-05-08

[36] Teradata Corporation: *Business Analytics, Hybrid Cloud & Consulting*. https://www.teradata.com/. – Last accessed: 2018-04-22

[37] The Gephi Consortium: *Gephi - The Open Graph Viz Platform*. https://gephi.org/. – Last accessed: 2018-04-05

[38] The Wall Street Journal: *Drunk on the Train Platform? These Cameras Can Tell*. https://blogs.wsj.com/japanrealtime/2015/08/13/drunk-on-the-train-platform-these-cameras-can-tell/. – Last accessed: 2017-05-15

[39] Trunomi: *EU GDPR Information Portal*. https://www.eugdpr.org/. – Last accessed: 2018-05-07

[40] U.S. General Services Administration: *Data.gov*. https://www.data.gov/. – Last accessed: 2018-05-06

[41] W3C: *RDF - Semantic Web Standards*. https://www.w3.org/RDF/. – Last accessed: 2018-04-25

[42] Wikipedia: *Data profiling*. https://en.wikipedia.org/wiki/Data_profiling. – Last accessed: 2018-01-10

[43] Wikipedia: *Profiling*. https://en.wikipedia.org/wiki/Profiling. – Last accessed: 2017-04-27