The VLDB Journal (2015) 24:557–581
DOI 10.1007/s00778-015-0389-y

REGULAR PAPER

Profiling relational data: a survey

Ziawasch Abedjan[1] · Lukasz Golab[2] · Felix Naumann[3]

Received: 1 August 2014 / Revised: 5 May 2015 / Accepted: 13 May 2015 / Published online: 2 June 2015

Abstract Profiling data to determine metadata about a given dataset is an important and frequent activity of any IT professional and researcher and is necessary for various use-cases. It encompasses a vast array of methods to examine datasets and produce metadata. Among the simpler results are statistics, such as the number of null values and distinct values in a column, its data type, or the most frequent patterns of its data values. Metadata that are more difficult to compute involve multiple columns, namely correlations, unique column combinations, functional dependencies, and inclusion dependencies. Further techniques detect conditional properties of the dataset at hand. This survey provides a classification of data profiling tasks and comprehensively reviews the state of the art for each class. In addition, we review data profiling tools and systems from research and industry. We conclude with an outlook on the future of data profiling beyond traditional profiling tasks and beyond relational databases.

✉ Felix Naumann
felix.naumann@hpi.de

Ziawasch Abedjan
abedjan@csail.mit.edu

Lukasz Golab
lgolab@uwaterloo.ca

1 MIT CSAIL, Cambridge, MA, USA

2 University of Waterloo, Waterloo, Canada

3 Hasso Plattner Institute, Potsdam, Germany

1 Data profiling: finding metadata

Data profiling is the set of activities and processes to determine the metadata about a given dataset. Profiling data is an important and frequent activity of any IT professional and researcher. We can safely assume that any reader of this article has engaged in the activity of data profiling, at least by eye-balling spreadsheets, database tables, XML files, etc. Possibly, more advanced techniques were used, such as keyword searching in datasets, writing structured queries, or even using dedicated data profiling tools.

Johnson gives the following definition: “Data profiling refers to the activity of creating small but informative summaries of a database” [79]. Data profiling encompasses a vast array of methods to examine datasets and produce metadata. Among the simpler results are statistics, such as the number of null values and distinct values in a column, its data type, or the most frequent patterns of its data values. Metadata that are more difficult to compute involve multiple columns, such as inclusion dependencies or functional dependencies. Also of practical interest are approximate versions of these dependencies, in particular because they are typically more efficient to compute. In this survey we preclude these and concentrate on exact methods.

Like many data management tasks, data profiling faces three challenges: *(i)* managing the input, *(ii)* performing the computation, and *(iii)* managing the output. Apart from typical data formatting issues, the first challenge addresses the problem of specifying the expected outcome, i.e., determining which profiling tasks to execute on which parts of the data. In fact, many tools require a precise specification of what to inspect. Other approaches are more open and perform a wider range of tasks, discovering all metadata automatically.

The second challenge is the main focus of this survey and that of most research in the area of data profiling: The computational complexity of data profiling algorithms depends on the number or rows, with a sort being a typical operation, but also on the number of columns. Many tasks need to inspect all column combinations, i.e., they are exponential in the number of columns. In addition, the scalability of data profiling methods is important, as the ever-growing data volumes demand disk-based and distributed processing.

The third challenge is arguably the most difficult, namely meaningfully interpreting the data profiling results. Obviously, any discovered metadata refer only to the given data instance and cannot be used to derive schematic/semantic properties with certainty, such as value domains, primary keys, or foreign key relationships. Thus, profiling results need interpretation, which is usually performed by database and domain experts.

Tools and algorithms have tackled these challenges in different ways. First, many rely on the capabilities of the underlying DBMS, as many profiling tasks can be expressed as SQL queries. Second, many have developed innovative ways to handle the individual challenges, for instance using indexing schemes, parallel processing, and reusing intermediate results. Third, several methods have been proposed that deliver only approximate results for various profiling tasks, for instance by profiling samples. Finally, users may be asked to narrow down the discovery process to certain columns or tables. For instance, there are tools that verify inclusion dependencies on user-suggested pairs of columns, but cannot automatically check inclusion between all pairs of columns or column sets.

Systematic data profiling, i.e., profiling beyond the occasional exploratory SQL query or spreadsheet browsing, is usually performed with dedicated tools or components, such as IBM's Information Analyzer, Microsoft's SQL Server Integration Services (SSIS), or Informatica's Data Explorer.[1] These approaches follow the same general procedure: A user specifies the data to be profiled and selects the types of metadata to be generated. Next, the tool computes the metadata in batch mode, using SQL queries and/or specialized algorithms. Depending on the volume of the data and the selected profiling results, this step can last minutes to hours. Results are usually displayed in a vast collection of tabs, tables, charts, and other visualizations to be explored by the user. Typically, discoveries can then be translated to constraints or rules that are then enforced in a subsequent cleansing/integration phase. For instance, after discovering that the most frequent pattern for phone numbers is `(ddd)ddd-dddd`, this pattern can be promoted to a *rule* stating that all phone numbers must be formatted accordingly. Most data cleansing tools can then either transform differently formatted numbers or mark them as improper.

[1] See Sect. 6 for a more comprehensive list of tools.

We focus our discussion on relational data, the predominant format of traditional data profiling methods, but we do cover data profiling for other data models in Sect. 7.2.

1.1 Use-cases for data profiling

Data profiling has many traditional use-cases, including the data exploration, data cleansing, and data integration scenarios. Statistics about data are also useful in query optimization. Finally we describe several domain-specific use-cases, such as scientific data management and big data analytics.

Data exploration Database administrators, researchers, and developers are often confronted with new datasets, about which they know nothing. Examples include data files downloaded from the Web, old database dumps, or newly gained access to some DBMS. In many cases, such data have no known schema, no or old documentation, etc. Even if a formal schema is specified, it might be incomplete, for instance specifying only the primary keys but no foreign keys. A natural first step is to understand how the data are structured, what they are about, and how much of them there are.

Such manual data exploration, or data gazing[2], can and should be supported with data profiling techniques. Simple, ad hoc SQL queries can reveal some insight, such as the number of distinct values, but more sophisticated methods are needed to efficiently and systematically discover metadata. Furthermore, we cannot always expect an SQL expert as the explorer, but rather "data enthusiasts" without formal computer science training [68]. Thus, automated data profiling is needed to provide a basis for further analysis. Morton et al. [107] recognize that a key challenge is overcoming the current assumption of data exploration tools that data are "clean and in a well-structured relational format." Often data cannot be analyzed and visualized as is.

Database management A basic form of data profiling is the analysis of individual columns in a given table. Typically, the generated metadata include various counts, such as the number of values, the number of unique values, and the number of non-null values. These metadata are often part of the basic statistics gathered by a DBMS. An optimizer uses them to estimate the selectivity of operators and perform other optimization steps. Mannino et al. [99] give a survey of statistics collection and its relationship to database optimization. More advanced techniques use histograms of value distributions, functional dependencies, and unique column combinations to optimize range queries [118] or for dynamic reoptimization [80].

[2] "Data gazing involves looking at the data and trying to reconstruct a story behind these data. [...] Data gazing mostly uses deduction and common sense." [104]

Database reverse engineering Given a "bare" database instance, the task of schema and database reverse engineering is to identify its relations and attributes, as well as domain semantics, such as foreign keys and cardinalities [103,116]. Hainaut et al. [66] call these metadata "implicit constructs," i.e., those that are not explicitly specified by DDL statements. However, possible sources for reverse engineering are DDL statements, data instances, data dictionaries, etc. The result of reverse engineering might be an entity-relationship model or a logical schema to assist experts in maintaining, integrating, and querying the database.

Data integration Often, the datasets to be integrated are unfamiliar and the integration expert wants to explore the datasets first: How large are they? What data types are needed? What are the semantics of columns and tables? Are there dependencies between tables and among databases?, etc. The vast abundance of (linked) *open data* and the desire and potential to integrate them with local data has amplified this need.

A concrete use-case for data profiling is that of *schema matching*, i.e., finding semantically correct correspondences between elements of two schemata [44]. Many schema matching systems perform data profiling to create attribute features, such as data type, average value length, and patterns, to compare feature vectors and align those attributes with the best matching ones [98,109].

Scientific data management and integration have created additional motivation for efficient and effective data profiling: When importing raw data, e.g., from scientific experiments or extracted from the Web, into a DBMS, it is often necessary and useful to profile the data and then devise an adequate schema. In many cases, scientific data are produced by non-database experts and without the intention to enable integration. Thus, they often come with no adequate schematic information, such as data types, keys, or foreign keys.

Apart from exploring individual sources, data profiling can also reveal how and how well two datasets can be integrated. For instance, inclusion dependencies across tables from different sources suggest which tables might reasonably be combined with a join operation. Additionally, specialized data profiling techniques can reveal how much two relations overlap in their intent and extent. We discuss these challenges in Sect. 7.1.

Data quality / data cleansing The need to profile a new or unfamiliar set of data arises in many situations, in general to prepare for some subsequent task. A typical use-case is profiling data to prepare a *data cleansing* process. Commercial data profiling tools are usually bundled with corresponding data quality / data cleansing software.

Profiling as a data quality assessment tool reveals data errors, such as inconsistent formatting within a column, missing values, or outliers. Profiling results can also be used to measure and monitor the general quality of a dataset, for instance by determining the number of records that do not conform to previously established constraints [81,117]. Generated constraints and dependencies also allow for rule-based data imputation.

Big data analytics "Big data," with its high volume, high velocity, and high variety [90], are data that cannot be managed with traditional techniques. Thus, data profiling gains a new importance. Fetching, storing, querying, and integrating big data are expensive, despite many modern technologies: Before exposing an infrastructure to Twitter's firehose, it might be worthwhile to know about properties of the data one is receiving; before downloading significant parts of the linked data cloud, some prior sense of the integration effort is needed; before augmenting a warehouse with text mining results an understanding of its data quality is required. In this context, leading researchers have noted "*If we just have a bunch of datasets in a repository, it is unlikely anyone will ever be able to find, let alone reuse, any of these data. With adequate metadata, there is some hope, but even so, challenges will remain*[...] [7]."

Many big data and related data science scenarios call for data mining and machine learning techniques to explore and mine data. Again, data profiling is an important preparatory task to determine which data to mine, how to import it into the various tools, and how to interpret the results [120].

Further use-cases Knowledge about data types, keys, foreign keys, and other constraints supports data modeling and helps keep data consistent, improves query optimization, and reaps all the other benefits of structured data management. Others have mentioned query formulation and indexing [126] and scientific discovery [75] as further motivation for data profiling. Also, compression techniques internally perform basic data profiling to optimize the compression ratio. Finally, the areas of data governance and data life-cycle management are becoming more and more relevant to businesses trying to adhere to regulations and code. Especially concerned are financial institutions and health care organizations. Again, data profiling can help ascertain which actions to take on which data.

1.2 Article overview and contributions

Data profiling is an important and practical topic that is closely connected to several other data management areas. It is also a timely topic and is becoming increasingly important given the recent trends in data science and big data analytics [108]. While it may not yet be a mainstream term in the database community, there already exists a large body of work that directly and indirectly addresses various aspects of data profiling. The goal of this survey is to classify and describe this body of work and illustrate its relevance to database research and practice. We also show that data profiling is far from a "done deal" and identify several promising directions for future work in this area.

The remainder of this paper is organized as follows. In Sect. 2, we outline and define data profiling based on a new taxonomy of profiling tasks. Sections 3, 4, and 5 survey the state of the art of the three main research areas in data profiling: analysis of individual columns, analysis of multiple columns, and detection of dependencies between columns, respectively. Section 6 surveys data profiling tools from research and industry. We provide an outlook of data profiling challenges in Sect. 7 and conclude this survey in Sect. 8.

2 Profiling tasks

This section presents a classification of data profiling tasks. Figure 1 shows our classification, which includes single-column tasks, multi-column tasks and dependency detection. While dependency detection falls under multi-column profiling, we chose to assign a separate profiling class to this large, complex, and important set of tasks. The classes are discussed in the following subsections. We also highlight additional dimensions of data profiling, such as the type of storage, the approximation of profiling results, as well as the relationship between data profiling and data mining.

Collectively, a set of results of these tasks is called the *data profile* or *database profile*. In general, we assume the dataset itself as our only input, i.e., we cannot rely on query logs, schema, documentation.

2.1 Single-column profiling

A basic form of data profiling is the analysis of individual columns in a given table. Typically, the generated metadata comprise various counts, such as the number of values, the number of unique values, and the number of non-null values. These metadata are often part of the basic statistics gathered by the DBMS. In addition, the maximum and minimum values are discovered and the data type is derived (usually restricted to string versus numeric versus date). More advanced techniques create histograms of value distributions and identify typical patterns in the data values in the form of regular expressions [122]. Data profiling tools display such results and can suggest actions, such as declaring a column with only unique values to be a key candidate or suggesting to enforce the most frequent patterns. As another exemplary

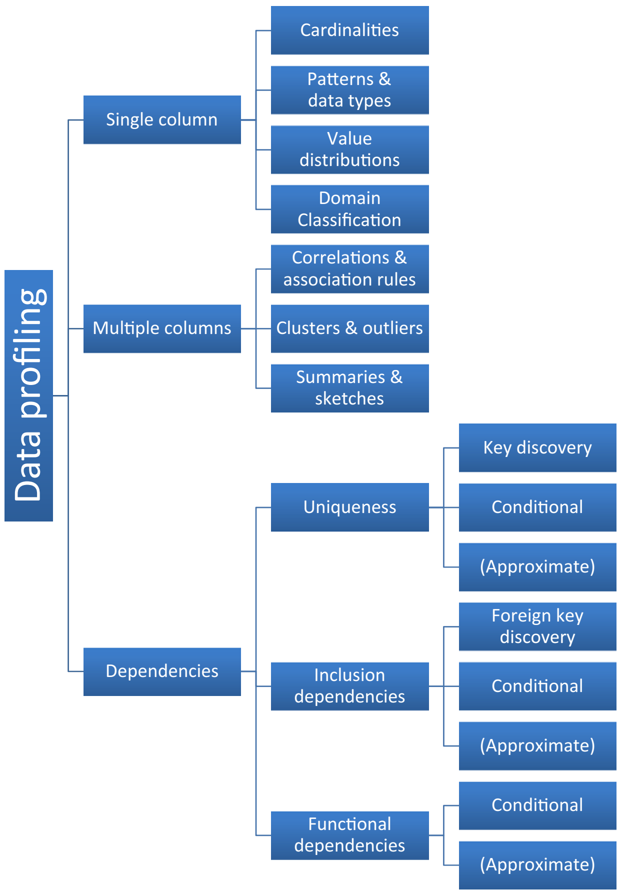

Fig. 1 A classification of traditional data profiling tasks

use-case, query optimizers in database management systems also make heavy use of such statistics to estimate the cost of an execution plan.

Table 1 lists the possible and typical metadata as a result of single-column data profiling. Some tasks are self-evident while others deserve more explanation. In Sect. 3, we elaborate on the more interesting tasks, their implementation, and their use.

2.2 Multi-column profiling

The second class of profiling tasks covers multiple columns simultaneously. Multi-column profiling generalizes profiling tasks on single columns to multiple columns and also identifies intervalue dependencies and column similarities. One task is to identify correlations between values through frequent patterns or association rules. Furthermore, clustering approaches that consume values of multiple columns as features allow for the discovery of coherent subsets of data records and outliers. Similarly, generating summaries and sketches of large datasets relates to profiling values across columns.

Table 1 Overview of selected single-column profiling tasks (see Sect. 3 for details)

Category	Task	Description
Cardinalities	num-rows	Number of rows
	value length	Measurements of value lengths (minimum, maximum, median, and average)
	null values	Number or percentage of null values
	distinct	Number of distinct values; sometimes called "cardinality"
	uniqueness	Number of distinct values divided by the number of rows
Value distributions	histogram	Frequency histograms (equi-width, equi-depth, etc.)
	constancy	Frequency of most frequent value divided by number of rows
	quartiles	Three points that divide the (numeric) values into four equal groups
	first digit	Distribution of first digit in numeric values; to check Benford's law
Patterns, data types, and domains	basic type	Generic data type, such as numeric, alphabetic, alphanumeric, date, time
	data type	Concrete DBMS-specific data type, such as varchar, timestamp.
	size	Maximum number of digits in numeric values
	decimals	Maximum number of decimals in numeric values
	patterns	Histogram of value patterns (Aa9…)
	data class	Semantic, generic data type, such as code, indicator, text, date/time, quantity, identifier
	domain	Classification of semantic domain, such as credit card, first name, city, phenotype

Such metadata are useful in many applications, such as data exploration and analytics. Outlier detection is used in data cleansing applications, where outliers may indicate incorrect data values.

Section 4 describes these tasks and techniques in more detail. It comprises multi-column profiling tasks that generate metadata on horizontal partitions of the data, such as values and records, instead vertical partitions, such as columns and column groups. Although the discovery of column dependencies, such as key or functional dependency discovery, also relates to multi-column profiling, we dedicate a separate section to dependency discovery as described next.

2.3 Dependencies

Dependencies are metadata that describe relationships among columns. The difficulties of automatically detecting such dependencies in a given dataset are twofold: First, pairs of columns or larger column sets must be examined, and second, the chance existence of a dependency in the data at hand does not imply that this dependency is meaningful. While much research has been invested in addressing the first challenge and is the focus of this survey, there is less work on semantically interpreting the profiling results.

A common goal of data profiling is to identify suitable keys for a given table. Thus, the discovery of *unique column combinations*, i.e., sets of columns whose values uniquely identify rows, is an important data profiling task [70]. Once unique column combinations have been discovered, a second step is to identify among them the intended primary key of a relation.

A frequent real-world use-case of multi-column profiling is the discovery of foreign keys [96,123] with the help of inclusion dependencies [14,100]. An inclusion dependency states that all values or value combinations from one set of columns also appear in the other set of columns—a prerequisite for a foreign key.

Another form of dependency that is also relevant for data quality is the functional dependency (FD). A functional dependency states that values in one set of columns functionally determine the value of another column. Again, much research has been performed to automatically detect FDs [75,139]. Section 5 surveys dependency discovery algorithms in detail.

Dependencies have many applications: An obvious use-case for functional dependencies is schema normalization. Inclusion dependencies can suggest how to join two relations, possibly across data sources. Their conditional counterparts help explore the data by focusing on certain parts of the dataset.

2.4 Conditional, partial, and approximate solutions

Real datasets usually contain exceptions to rules. To account for this, dependencies and other constraints detected by data profiling can be relaxed. We describe two relaxations below: partial and approximate.

Partial dependencies hold for only a subset of the records, for instance, for 95 % of the records or for all but 10 records. Such dependencies are especially valuable in data cleansing scenarios: They are patterns that hold for almost all records and thus should probably hold for *all* records if the data were clean. Violating records can be extracted and cleansed [129].

Once a partial dependency has been detected, it is interesting to characterize for which records it holds, i.e., if we can find a condition that selects precisely those records. *Conditional dependencies* can specify such conditions. For instance, a conditional unique column combination might state that the column street is unique for all records with city = 'NY.' Conditional inclusion dependencies (CINDs) were proposed by Bravo et al. for data cleaning and contextual schema matching [19]. Conditional functional dependencies (CFDs) were introduced in [46], also for data cleaning.

Approximate dependencies and other constraints are unconditional statements, but are not guaranteed to hold for the entire relation. Such dependencies are often discovered using sampling [76] or other summarization techniques [31]. Their approximate nature is often sufficient for certain tasks, and approximate dependencies can be used as input to the more rigorous task of detecting true dependencies. This survey does not discuss such approximation techniques.

2.5 Types of storage

Data profiling tasks are applicable to a wide range of situations in which data are provided in various forms. For instance, most commercial profiling tools assume that data reside in a relational database, make use of SQL queries and indexes. In other situations, for instance, a csv file is provided and a data profiling method needs to create its own data structures in memory or on disk. And finally, there are situations in which a mixed approach is useful: Data that were originally in the database are read once and processed further outside the database.

The discussion and distinction of such different situations is relevant when evaluating the performance of data profiling algorithms and tools. Can we assume that data are already loaded into main memory? Can we assume the presence of indices? Are profiling results, which can be quite voluminous, written to disk? Fair comparisons need to establish a level playing field with same assumptions about data storage.

2.6 Data profiling versus data mining

A clear, well-defined, and accepted distinction between data profiling and data mining does not exist. Two criteria are conceivable:

1. Distinction by the object of analysis: instance versus schema or columns versus rows
2. Distinction by the goal of the task: description of existing data versus new insights beyond existing data.

Following the first criterion, Rahm and Do distinguish data profiling from data mining by the number of columns that are examined: "Data profiling focuses on the instance analysis of individual attributes. [...] Data mining helps discover specific data patterns in large datasets, e.g., relationships holding between several attributes" [121]. While this distinction is well defined, we believe several tasks, such as IND or FD detection, belong to data profiling, even if they discover relationships between multiple columns.

We believe a different distinction along both criteria is more useful: Data profiling gathers technical metadata to support data management; data mining and data analytics discovers non-obvious results to support business management with new insights. While data profiling focuses mainly on columns, some data mining tasks, such as rule discovery or clustering, may also be used for identifying interesting characteristics of a dataset. Others, such as recommendation or classification, are not related to data profiling.

With this distinction, we concentrate on data profiling and put aside the broad area of data mining, which has already received unifying treatment in numerous textbooks and surveys. However, in Sect. 4, we address the subset of unsupervised mining approaches that can be applied on unknown data to generate metadata and hence serves the purpose of data profiling.

Classifications of data mining tasks include an overview by Chen et al., who distinguish the kinds of databases (relational, OO, temporal, etc.), the kinds of knowledge to be mined (association rules, clustering, deviation analysis, etc.), and the kinds of techniques to be used [130]. We make a similar distinction in this survey. In particular, we distinguish the different classes of data profiling tasks and then examine various techniques to perform them. We discuss profiling non-relational data in Sect. 7.

2.7 Summary

We summarize this section by connecting the various data profiling tasks with the use-cases mentioned in the introduction. Conceivably, any task can be useful for any use-case, depending on the context, the properties of the data at hand,

Table 2 Data profiling tasks and their primary use-cases

	Database management	Data integration	Data cleansing	Database reverse engineering	Data exploration	Data analytics	Scientific data management
Single-column							
Cardinalities	✓				✓	✓	
Patterns and data types		✓	✓	✓			
Value distributions	✓		✓		✓	✓	
Domain classification		✓	✓	✓			✓
Multi-column							
Correlations					✓	✓	✓
Association rules						✓	✓
Clustering		✓			✓	✓	✓
Outliers			✓				✓
Summaries and sketches			✓		✓	✓	
Dependencies							
Unique column combinations	✓		✓	✓			
Inclusion dependencies	✓	✓		✓	✓		✓
Conditional inclusion dependencies		✓			✓	✓	✓
Functional dependencies	✓		✓	✓	✓		
Conditional functional dependencies			✓		✓	✓	

etc. Table 2 lists the profiling tasks and their primary use-cases.

3 Column analysis

The analysis of the values of individual columns is usually a straightforward task. Table 1 lists the typical metadata that can determined for a given column. The following sections describe each category of tasks in more detail, mentioning possible uses of the respective results. In [104], a book addressing practitioners, several of these tasks are discussed in more detail.

3.1 Cardinalities

Cardinalities or counts of values in a column are the most basic form of metadata. The number of rows in a table (num-rows) reflects how many entities (e.g., customers, orders, items) are represented in the data, and it is relevant to data management systems, for instance to estimate query costs or to assign storage space.

Information about the length of values in terms of characters (value length), including the length of the longest and shortest value and the average length, is useful for schema reverse engineering (e.g., to determine tight data type bounds), outlier detection (e.g., single-character first names), and formatting (dates have the same min-, max- and average length).

The number of empty cells, i.e., cells with null values or empty strings (null values), indicates the (in-)completeness of a column. The number of distinct values (distinct) allows query optimizers to estimate selectivity of selection or join operations: The more distinct values there are, the more selective such operations are. To users, this number can indicate a candidate key by comparing it with the number of rows. Alternatively, this number simply illustrates how many different values are present (e.g., how many customers have ordered something or how many cities appear in an address table).

Determining the number of rows, metadata about value lengths, and the number of null values is straightforward and can be performed in a single pass over the data. Determining the number of *distinct values* is more involved: Either hashing or sorting all values is necessary. When hashing, the number of non-empty buckets must be counted, taking into account hash collisions, which further add to the count. When sorting, a pass through the sorted data counts the number of values, where groups of same values are counted only once.

From the number of distinct values the *uniqueness* can be calculated, which is typically defined as the number of unique values divided by the number of rows. Note that the number of distinct values can also be estimated using the minHash technique discussed in Sect. 4.3.

Apart from determining the exact number of distinct values, query optimization is a strong incentive to *estimate* those counts in order to predict query execution plan costs without actually reading the entire data. Because approximate profiling is not the focus of this survey, we give only two exemplary pointers. Haas et al. [65] base their estimation on data samples and describe and empirically compare various estimators from the literature. Other works do scan the entire data but use only a small amount of memory to hash the values and estimate the number of distinct values, an early example being [11].

3.2 Value distribution

Value distributions are more fine-grained cardinalities, namely the cardinalities of groups of values. *Histograms* are among the most common profiling results. A histogram stores frequencies of values within well-defined groups, usually by dividing the ordered set of values into a fixed set of buckets. The buckets of equi-width histograms span value ranges of same length, while the buckets of equi-depth (or equi-height) histograms each represent the same number of value occurrences. A common special case of an equi-depth histogram is dividing the data into four *quartiles*. A more general concept is *biased histograms*, which can adapt their accuracy for different regions[33]. Histograms are used for database optimization as a rough probability distribution to avoid a uniform distribution assumption and thus provide better cardinality estimations [77]. In addition, histograms are interpretable by humans, as their visual representation is easy to comprehend.

The *constancy* of a column is defined as the ratio of the frequency of the most frequent value (possibly a pre-defined default value) and the overall number of values. It thus represents the proportion of some constant value compared with the entire column.

A particularly interesting distribution is the first digit distribution for numeric values. Benford's law [15] states that in naturally occurring numbers the distribution of the first digit d of a number approximately follows $P(d) = \log_{10}(1 + \frac{1}{d})$. Thus, the 1 is expected to be the most frequent leading digit, followed by 2, etc. Benford and others have observed this behavior in many sets of numbers, such as molecular weights, building sizes, and electricity bills. In fact, the law has been used to uncover accounting fraud and other fraudulently created numbers.

Determining the above distributions usually involves a single pass over the column, except for equi-depth histograms (i.e., with fixed bucket sizes) and quartiles, which determine bucket boundaries through sorting. In the same manner or through hashing the most frequent value can be discovered to determine constancy.

Finally, many more things can be counted and aggregated in a column. For instance, some profiling tools and methods determine among others the frequency distribution of soundex code, n-grams, and others, the inverse frequency distribution, i.e., the distribution of the frequency distribution, or the entropy of the frequency distribution of the values in a column [82].

3.3 Types and patterns

The profiling tasks of this section are ordered by increasing semantic richness (see also Table 1). We start with the most simple observable properties, move on to specific patterns of the values of a column, and end with the semantic domain of a column.

Discovering the *basic type* of a column, i.e., classifying it as numeric, alphabetic, alphanumeric, date, or time, is fairly simple: The presence or absence of numeric and non-numeric characters already distinguishes the first three. The latter two can usually be recognized by the presence of numbers only within certain ranges, and by numbers separated in regular patterns by special symbols. Recognizing the actual data type, for instance among the SQL types, is similarly easy. In fact, data of many data types, such as timestamp, boolean, or int, must follow a fixed, sometimes DBMS-specific pattern. When classifying columns into data types, one should choose the most specific data type—in particular avoiding the catchalls char or varchar if possible. For the data types decimal, float, and double, one can additionally extract the maximum number of digits and decimals to determine the metadata size and decimals.

A common and useful data profiling result is the extraction of frequent *patterns* observed in the data of a column. Then, data that do not conform to such a pattern are likely erroneous or ill-formed. For instance, a pattern for phone numbers might be informally encoded as `+dd (ddd) ddd dddd` or as a simple regular expression `\(\d3\)\ − \d3\ − \d4)`.[3] A challenge when determining frequent patterns is to find a good balance between generality and specificity. The regular expression `.*` is the most general and matches any string. On the other hand, the expression `data` allows only that one single string. For the Potter's Wheel tool, Raman and Hellerstein [122] suggest finding the data pattern with the minimal description length (MDL). They model description length as a combination of precision, recall, and conciseness and provide an algorithm to enumerate all possible patterns. The RelIE system was designed

[3] A more detailed regular expression, taking into account different formatting options and different restrictions (e.g., phone numbers cannot begin with a 1), can easily reach 200 characters in length.

for information extraction from textual data [92]. It creates regular expressions based on training data with positive and negative examples by systematically, greedily transforming regular expressions. Finally, Fernau [51] provides a good characterization of the problem of learning regular expressions from data and presents a learning algorithm for the task. This work is also a good starting point for further reading

The semantic *domain* of a column describes not the syntax of its values but their meaning. While a regular expression might characterize a column, labeling it as "phone number" provides a concrete domain. Clearly, this task cannot be fully automated, but some work has been done for common-place domains about persons, places, etc. Zhang et al. take a first step by clustering columns that have the same meaning across the tables of a database [144], which they extend to the particularly difficult area of numeric values in [142]. In [133] the authors take the additional step of matching columns to pre-defined semantics from the person domain. Knowledge of the domain is not only of general data profiling interest, but also of particular interest to schema matching, i.e., the task of finding semantic correspondences between elements of different database schemata.

3.4 Data completeness

Explicit missing data are simple to characterize: For each column, we report the number of tuples with a null or a default value. However, datasets may contain *disguised* missing values. For example, Web forms often include fields whose values must be chosen from pull-down lists. The first value from the pull-down list may be pre-populated on the form, and some users may not replace it with a proper or correct value due to lack of time or privacy concerns. Specific examples include entering 99999 as the zip code of an address or leaving "Alabama" as the pre-populated state (in the US, Alabama is alphabetically the first state). Of course, for some records, Alabama may be the true state.

Detecting disguised default values is difficult. One heuristic solution is to examine each column at a time, and, for each possible value, compute the distribution of the other attribute values [74]. For example, if Alabama is indeed a disguised default value, we expect a large subset of tuples with state = `Alabama` (i.e., those whose true state is different) to form an unbiased sample of the whole relation.

Another instance in which profiling missing data is not trivial involves timestamped data, such as measurement or transaction data feeds. In some cases, tuples are expected to arrive regularly, e.g., in datacenter monitoring, every machine may be configured to report its CPU utilization every minute. However, measurements may be lost en route to the database, and overloaded or malfunctioning machines may not report any measurements at all. [60]. In contrast to detecting missing attribute values, here we are interested in estimating the number of missing tuples. Thus, the profiling task may be to single out the columns identified as being of type timestamp, and, for those that appear to be distributed uniformly across a range, infer the expected frequency of the underlying data source and estimate the number of missing tuples. Of course, some timestamp columns correspond to application timestamps with no expectation of regularity, rather than data arrival timestamps. For instance, in an online retailer database, order dates and delivery dates are generally not expected to be scattered uniformly over time.

4 Multi-column analysis

Profiling tasks over a single column can be generalized to projections of multiple columns. For example, there has been work on computing multi-dimensional histograms for query optimization [41,119]. Multi-column profiling also plays an important role in data cleansing, e.g., in assessing and explaining data glitches, which often occur in column combinations [40].

In the remainder of this section, we discuss statistical methods and data mining approaches for generating metadata based on co-occurrences and dependencies of values across attributes. We focus on correlation and rule mining approaches as well as unsupervised clustering and learning approaches; machine learning techniques that require training data or detailed knowledge of the data are beyond the scope of data profiling.

4.1 Correlations and association rules

Correlation analysis reveals related numeric columns, e.g., in an Employees table, age and salary may be correlated. A straightforward way to do this is to compute pairwise correlations among all pairs of columns. In addition to column-level correlations, value-level *associations* may provide useful data profiling information.

Traditionally, a common application of association rules has been to find items that tend to be purchased together based on point-of-sale transaction data. In these datasets, each row is a list of items purchased in a given transaction. An association rule {`bread`} → {`butter`}, for example, states that if a transaction includes bread, it is also likely to include butter, i.e., customers who buy bread also buy butter. A set of items is referred to as an *itemset*, and an association rule specifies an itemset on the left-hand side and another itemset on the right-hand side.

Algorithms for generating association rules from data decompose the problem into two steps [8]:

1. Discover all frequent itemsets, i.e., those whose frequencies in the dataset (i.e., their *support*) exceed some

threshold. For instance, the itemset {`bread`, `butter`} may appear in 800 out of a total of 50,000 transactions for a support of 1.6 %.

2. For each frequent itemset a, generate association rules of the form $l \rightarrow a - l$ with $l \subset a$, whose *confidence* exceeds some threshold. Confidence is defined as the frequency of a divided by the frequency of l, i.e., the conditional probability of l given $a - l$. For example, if the frequency of {`bread`, `butter`} is 800 and the frequency of {`bread`} alone is 1000, then the confidence of the association rule {`bread`} → {`butter`} is 0.8. That is, if bread is purchased, there is an 80 % chance that butter is also purchased in the same transaction.

In the context of relational data profiling, association rules denote relationships or patterns between attribute values among columns. Consider an Employees table with fields name, employee number, department, position, and allowance. We may find a frequent itemset of the form {department = `finance`, position = `assistant manager`, allowance = `$1000`} and a corresponding association rule of the form {department = `finance`, position = `assistant manager`} → {allowance = `$1000`}. This would be the case if most or all assistant managers in the finance department were assigned an allowance budget of $1000.

While the second step mentioned above is straightforward (generating association rules from frequent itemsets), the first step is computationally expensive due to the large number of possible frequent itemsets (or patterns of values) [72]. Popular algorithms for efficiently discovering frequent patterns include Apriori [8], Eclat [141], and FP-Growth [67].

The Apriori algorithm exploits the observation that all subsets of a frequent itemset must also be frequent. In the first iteration, Apriori finds all frequent itemsets of size one, i.e., those containing one item or one attribute value. In the next iteration, only the frequent itemsets of size one are expanded to find frequent itemsets of size two, and so on.

There are also several optimized versions of Apriori, such as DHP [115] and RARM [35]. FP-Growth discovers frequent itemsets without a candidate generation step. It transforms the database into an extended prefix tree of frequent patterns (FP-tree). The FP-Growth algorithm traverses the tree and generates frequent itemsets by pattern growth in a depth-first manner. Finally, Eclat is based on intersecting transaction-id (TID) sets of associated itemsets and is best suited for dealing with large frequent itemsets. Eclat's strategy for identifying frequent itemsets is similar to Apriori.

Negative correlation rules, i.e., those that identify attribute values that *do not* co-occur with other attribute values, may also be useful in data profiling to find anomalies and outliers [21]. However, discovering negative association rules is more difficult, because *infrequent* itemsets cannot be pruned in the same way as frequent itemsets, and therefore, novel pruning rules are required [135].

Finally, we note that in addition to using existing techniques, such as correlations and association rules for profiling, extensions have been proposed, such as discovering linear dependencies between columns [25].

However, in this approach, the user has to choose the subset of attributes to be analyzed. We discuss dependency discovery in more detail in Sect. 5.

4.2 Clustering and outlier detection

Another useful profiling task is to segment the records into homogeneous groups using a clustering algorithm; furthermore, records that do not fit into any cluster may be flagged as outliers. Cluster analysis can identify groups of similar records in a table, while outliers may indicate data quality problems. For example, Dasu and Johnson [36] cluster numeric columns and identify outliers in the data. Furthermore, based on the assumption that data glitches occur across attributes and not in isolation [16], statistical inference has been applied to measure glitch recovery in [39].

Another example of clustering in the context of data profiling is ProLOD++, which applies k-means clustering to RDF relations [1]. We refer the reader to surveys by Jain et al. [78] and Xu and Wunsch II [137] for more details on clustering algorithms for relational data.

4.3 Summaries and sketches

Besides clustering, another way to describe data is to create summaries or sketches [23]. This can be done by sampling or hashing data values to a smaller domain. Sketches have been widely applied to answering approximate queries, data stream processing and estimating join sizes [37,54,111]. Cormode et al. [31] give an overview of sketching and sampling for approximate query processing.

Another interesting task is to assess the similarity of two columns, which can be done using multi-column hashing techniques. The *Jaccard similarity* of two columns A and B is $|A \cap B|/|A \cup B|$, i.e., the number of distinct values they have in common divided by the total number of distinct values appearing in them. This gives the relative number of values that appear in both A and B. Since semantically similar values may have different formats, we can also compute the Jaccard similarity of the n-gram distributions in A and B. If the distinct value sets of columns A and B are not available, we can estimate the Jaccard similarity using their *MinHash signatures* [38].

Table 3 Dependency discovery algorithms

Dependency	**Algorithms**
Uniques	HCA [3], GORDIAN [126], DUCC [70], SWAN [5]
Functional dependencies	TANE [75], FUN [110], FD_Mine [139], Dep-Miner [95], FastFDs [136], FDEP [52], DFD[6]
Conditional functional dependencies	[24], [59], CTANE [47], CFUN [42], FACD [91], FastCFD [47]
Inclusion dependencies	[101], [87], SPIDER [14], ZigZag [102]
Conditional inclusion dependencies	[61], CINDERELLA [13], PLI [13]
Foreign keys	[123], [143]
Denial constraints	FastDC [29]
Differential dependencies	[128]
Sequential dependencies	[57]

5 Dependency detection

We now survey various formalisms for detecting dependencies among columns and algorithms for mining them from data, including keys and unique column combinations (Sect. 5.1), functional dependencies (Sect. 5.2), inclusion dependencies (Sect. 5.3), and other types of dependencies that are relevant to data profiling (Sect. 5.4). Table 3 lists the algorithms that are discussed.

We use the following symbols: R and S denote relational schemata, with r and s denoting instances of R and S, respectively. The number of columns in R is $|R|$ and the number of tuples in r is $|r|$. We refer to tuples of r and s as r_i and s_j, respectively. Subsets of columns are denoted by uppercase X, Y, Z (with $|X|$ denoting the number of columns in X) and individual columns by uppercase A, B, C. Furthermore, we define $\pi_X(r)$ and $\pi_A(r)$ as the projection of r on the attribute set X or attribute A, respectively; thus, $|\pi_X(r)|$ denotes the count of district combinations of the values of X appearing in r. Accordingly, $r_i[A]$ indicates the value of the attribute A of tuple r_i and $r_i[X] = \pi_X(r_i)$. We refer to an attribute value of a tuple as a *cell*.

The number of potential dependencies in r can be exponential in the number of attributes $|R|$; see Fig. 2 for an illustration of all possible subsets of the attributes in Table 4. This means that any dependency discovery algorithm has a worst-case exponential time complexity. There are two classes of heuristics that have appeared in the literature.

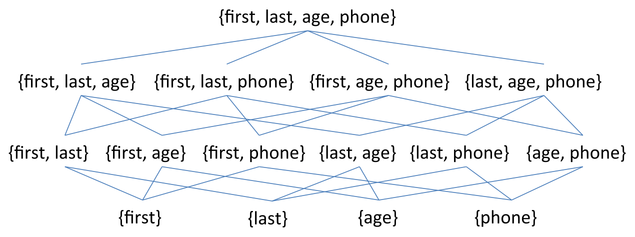

Fig. 2 Powerset lattice for the example Table 4

Table 4 Example dataset

Tuple id	First	Last	Age	Phone
1	Max	Payne	32	1234
2	Eve	Smith	24	5432
3	Eve	Payne	24	3333
4	Max	Payne	24	3333

Column-based or top-down approaches start with "small" dependencies (in terms of the number of attributes they reference) and work their way to larger dependencies, pruning candidates along the way whenever possible. Row-based or bottom-up approaches attempt to avoid repeated scanning of the entire relation during candidate generation. While these approaches cannot reduce the worst-case exponential complexity of dependency discovery, experimental studies have shown that column-based approaches work well on tables containing a very large number of rows and row-based approaches work well for wide tables [6,113]. For more details on the computational complexity of various FD and IND discovery algorithms, we refer the interested reader to [94].

5.1 Unique column combinations and keys

Given a relation R with instance r, a *unique column combination* (a "unique") is a set of columns $X \subseteq R$ whose projection on r contains only unique value combinations.

Definition 1 (*Unique*) A column combination $X \subseteq R$ is a *unique*, iff $\forall r_i, r_j \in r, i \neq j : r_i[X] \neq r_j[X]$.

Analogously, a set of columns $X \subseteq R$ is a *non-unique column combination* (a "non-unique"), iff its projection on r contains at least one duplicate value combination.

Definition 2 (*Non-unique*) A column combination $X \subseteq R$ is a *non-unique*, iff $\exists r_i, r_j \in r, i \neq j : r_i[X] = r_j[X]$.

Each superset of a unique is also unique while each subset of a non-unique is also a non-unique. Therefore, discovering all uniques and non-uniques can be reduced to the discovery of minimal uniques and maximal non-uniques:

Definition 3 (*Minimal Unique*) A column combination $X \subseteq R$ is a *minimal unique*, iff $\forall X' \subset X : X'$ is a non-unique.

Definition 4 (*Maximal Non-Unique*) A column combination $X \subseteq R$ is a *maximal non-unique*, iff $\forall X' \supset X : X'$ is a unique.

A *primary key* is a unique that was explicitly chosen to be the unique record identifier while designing the table schema. Since the discovered uniqueness constraints are only valid for a relational instance at a specific point of time, we refer to uniques and non-uniques instead of keys and non-keys. A further distinction can be made in terms of possible keys and certain keys when dealing with uncertain data and NULL values [86].

The problem of discovering a minimal unique of size $k \leq n$ is NP-complete [97]. To discover all minimal uniques and maximal non-uniques of a relational instance, in the worst case, one has to visit all subsets of the given relation, no matter the strategy (breadth-first or depth-first) or direction (bottom-up or top-down). Thus, the discovery of all minimal uniques and maximal non-uniques of a relational instance is an NP-hard problem and even the solution set can be exponential [64].

Given $|R|$, there can be $\binom{|R|}{\frac{|R|}{2}} \geq 2^{\frac{|R|}{2}}$ minimal uniques in the worst case, as all combinations of size $\frac{|R|}{2}$ can simultaneously be minimal uniques.

5.1.1 GORDIAN: *row-based discovery*

Row-based algorithms require multiple runs over all column combinations as more and more rows are considered. They benefit from the intuition that non-uniques can be detected without considering every row. A recursive unique discovery algorithm that works this way is GORDIAN [126]. The algorithm consists of three parts: *(i)* Pre-organize the data in form of a prefix tree, *(ii)* find maximal non-uniques by traversing the prefix tree, *(iii)* compute minimal uniques from maximal non-uniques.

The prefix tree is stored in main memory. Each level of the tree represents one column of the table, whereas each branch stands for one distinct tuple. Tuples that have the same values in their prefix share the corresponding branches. For example, all tuples that have the same value in the first column share the same node cells. The time to create the prefix tree depends on the number of rows; therefore, this can be a bottleneck for very large datasets.

The traversal of the tree is based on the cube operator [63], which computes aggregate functions on projected columns. Non-unique discovery is performed by a depth-first traversal of the tree for discovering maximum repeated branches, which constitute maximal non-uniques.

After discovering all maximal non-uniques, GORDIAN computes all minimal uniques by generating minimal combinations that are not covered by any of the maximal non-uniques. In [126] it is stated that this complementation step needs only quadratic time in the number of minimal uniques, but the presented algorithm implies cubic runtime: For each non-unique, the updated set of minimal uniques must be *simplified* by removing redundant uniques. This simplification requires quadratic runtime in the number of uniques. As the number of minimal uniques is bound linearly by the number s of maximal non-uniques, the runtime of the unique generation step is $O(s^3)$.

GORDIAN exploits the intuition that non-uniques can be discovered faster than uniques. Non-unique discovery can be aborted as soon as one repeated value is discovered among the projections. The prefix structure of the data facilitates this analysis. It is stated that the algorithm is polynomial in the number of tuples for data with a Zipfian distribution of values. Nevertheless, in the worst case, GORDIAN has exponential runtime.

The generation of minimal uniques from maximal non-uniques can be a bottleneck if there are many maximal non-uniques. Experiments showed that in most cases the unique generation dominates the runtime [3]. Furthermore, the approach is limited by the available main memory. Although data may be compressed because of the prefix structure of the tree, the amount of processed data may still be too large to fit in main memory.

5.1.2 *Column-based traversal of the column lattice*

The problem of finding minimal uniques is comparable to the problem of finding frequent itemsets [8]. The well-known Apriori approach is applicable to minimal unique discovery, working bottom-up as well as top-down. With regard to the powerset lattice of a relational schema, the Apriori algorithms generate all relevant column combinations of a certain size and verify those at once. Figure 2 illustrates the powerset lattice for the running example in Table 4. The effectiveness and theoretical background of those algorithms is discussed by Giannela and Wyss [55]. They presented three breadth-first traversal strategies: a bottom-up, a top-down, and a hybrid traversal strategy.

Bottom-up unique discovery traverses the powerset lattice of the schema R from the bottom, beginning with all 1-*combinations* toward the top of the lattice, which is the $|R|$-*combination*. The prefixed number k of *k-combination* indicates the size of the combination. The same notation applies for *k-candidates*, *k-uniques*, and *k-non-uniques*. To generate the set of *2-candidates*, we generate all pairs of

1-non-uniques. *k-candidates* with $k > 2$ are generated by extending the $(k-1)$*-non-uniques* by another non-unique column. After the candidate generation, each candidate is checked for uniqueness. If it is identified as a non-unique, the *k-candidate* is added to the list of *k-non-uniques*.

If the candidate is verified as unique, its minimality has to be checked. The algorithm terminates when $k = |1\text{-}non\text{-}uniques|$. A disadvantage of this candidate generation technique is that redundant uniques and duplicate candidates are generated and tested.

The Apriori idea can also be applied to the top-down approach. Having the set of identified *k-uniques*, one has to verify whether the uniques are minimal. Therefore, for each *k-unique*, all possible $(k-1)$*-subsets* have to be generated and verified. The hybrid approach generates the kth and $(n-k)$th levels simultaneously. Experiments have shown that in most datasets, uniques usually occur in the lower levels of the lattice, which favors bottom-up traversal [3].

HCA is an improved version of the bottom-up Apriori technique [3]. HCA optimizes the candidate generation step, applies statistical pruning and considers functional dependencies that have been inferred on the fly. In terms of candidate generation, HCA applies the optimized join that was introduced for frequent itemset mining [8]. HCA generates candidates by combining only $(k-1)$*-non-uniques* that share the first $k-2$ elements. If no such two non-uniques exist, no candidates are generated and the algorithm terminates before reaching the last level of the powerset lattice. Further pruning can be achieved by considering value histograms and distinct counts that can be retrieved on the fly in previous levels. For example, consider the *1-non-uniques* last and age from Table 4. The column combination {last,age} cannot be a unique based on the value distributions. While the value "`Payne`" occurs three times in last, the column age contains only two distinct values. That means at least two of the rows containing the value "`Payne`" also have a duplicate value in the age column. Using the count distinct values, HCA detects functional dependencies on the fly and leverages them to avoid unnecessary uniqueness checks.

While HCA improves existing bottom-up approaches, it does not perform the early identification of non-uniques in a row-based manner done by GORDIAN. Thus, GORDIAN is faster on datasets with many non-uniques, but HCA works better on datasets with many minimal uniques.

5.1.3 DUCC: traversing the lattice via random walk

While the breadth-first approach for discovering minimal uniques gives the most pruning, a depth-first approach might work well if there are relatively few minimal uniques that are scattered on different levels of the powerset lattice. Depth-first detection of unique column combinations resembles the problem of identifying the most promising paths through the lattice to discover existing minimal uniques and avoid unnecessary uniqueness checks. DUCC is a depth-first approach that traverses the lattice back and forth based on the uniqueness of combinations [70]. Following a random walk principle by randomly adding columns to non-uniques and removing columns from uniques, DUCC traverses the lattice in a manner that resembles the border between uniques and non-uniques in the powerset lattice of the schema.

DUCC starts with a seed set of *2-non-uniques* and picks a seed at random. Each k-combination is checked using the superset/subset relations and pruned if any of them subsumes the current combination. If no previously identified combination subsumes the current combination DUCC performs uniqueness verification. Depending on the verification, DUCC proceeds with an unchecked $(k-1)$-subset or $(k-1)$-superset of the current k-combination. If no seeds are available, it checks whether the set of discovered minimal uniques and maximal non-uniques correctly complement each other. If so, DUCC terminates; otherwise, a new seed set is generated by complementation.

DUCC also optimizes the verification of minimal uniques by using a position list index (PLI) representation of values of a column combination. In this index, each position list contains the tuple ids that correspond to the same value combination. Position lists with only one tuple id can be discarded, so that the position list index of a unique contains no position lists. To obtain the PLI of a column combination, the position lists in PLIs of all contained columns have to be cross-intersected. In fact, DUCC intersects two PLIs in a similar way in which a hash join operator would join two relations. As a result of using PLIs, DUCC can also apply row-based pruning, because the total number of positions decreases with the size of column combinations. Intuitively, combining columns makes the contained combination values more specific and therefore more likely to be distinct.

DUCC has been experimentally compared to HCA, a column-based approach, and GORDIAN, a row-based unique discovery algorithm. Since DUCC combines row-based and column-based pruning, it performs significantly better [70]. Experiments on smaller datasets showed that while HCA outperforms GORDIAN on low-dimensional data with many uniques, GORDIAN outperforms HCA on datasets with many attributes but few uniques [3].

Furthermore the random walk strategy allows a distributed application of DUCC for better scalability.

5.1.4 SWAN: an incremental approach

SWAN maintains a set of indexes to efficiently find the new sets of minimal uniques and maximal non-uniques after inserting or deleting tuples [5]. SWAN builds such indexes based on existing minimal uniques and maximal non-uniques in a way that avoids a full table scan. SWAN consists of two main components: the *Inserts Handler* and the *Deletes Handler*. The Inserts Handler takes as input a set of inserted tuples, checks all minimal uniques for uniqueness, finds the new sets of minimal uniques and maximal non-uniques, and updates the repository of minimal uniques and maximal non-uniques accordingly. Similarly, the Deletes Handler takes as input a set of deleted tuples, searches for duplicates in all maximal non-uniques, finds the new sets of minimal uniques and maximal non-uniques, and updates the repository accordingly.

5.2 Functional dependencies

A *functional dependency* (FD) 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, $r_i[A] = r_j[A]$. That is, any two tuples that agree on X must also agree on A. We refer to X as the left-hand side (LHS) and A as the right-hand side (RHS). Given r, we are interested in finding all non-trivial and minimal FDs $X \rightarrow A$ that hold on r, with non-trivial meaning $A \cap X = \emptyset$ and minimal meaning that there must not be any FD $Y \rightarrow A$ for any $Y \subset X$. A naive solution to the FD discovery problem is as follows.

```
For each possible RHS $A$
  For each possible LHS $X \in R \backslash A$
    For each pair of tuples $r_i$ and $r_j$
      If $r_i[X] = r_j[X]$ and $r_i[A] \neq r_j[A]$ Break
  Return $X \rightarrow A$
```

This algorithm is prohibitively expensive: For each of the $|R|$ possibilities for the RHS, it tests $2^{(|R|-1)}$ possibilities for the LHS, each time having to scan r multiple times to compare all pairs of tuples. However, notice that for $X \rightarrow A$ to hold, the number of distinct values of X must be the same as the number of distinct values of XA—otherwise at least one combination of values of X that is associated with more than one value of A, thereby breaking the FD [75]. Thus, if we pre-compute the number of distinct values of each combination of one or more columns, the algorithm simplifies to:

```
For each possible RHS $A$
  For each possible LHS $X \in R \backslash A$
    If $|\pi_X(r)| = |\pi_{XA}(r)|$
      Return $X \rightarrow A$
```

Recall Table 4. We have $|\pi_{\text{phone}}(r)| = |\pi_{\text{age,phone}}(r)| = |\pi_{\text{last,phone}}(r)|$. Thus, phone $\rightarrow$ age and phone $\rightarrow$ last hold. Furthermore, $|\pi_{\text{last,age}}(r)| = |\pi_{\text{last,age,phone}}(r)|$, implying {last,age} $\rightarrow$ phone.

The above algorithm is still inefficient due to the need to compute distinct value counts and test all possible column combinations. As was the case with unique discovery, FD discovery algorithms employ row-based (bottom-up) and column-based (top-down) optimizations, as discussed below. Figure 3 lists the algorithms that are discussed, along with their extensions to conditional FD discovery, which are covered in Sect. 5.2.4. An extensive experimental evaluation of various FD discovery algorithms on different datasets, scaling in both the number of rows and the number of columns, is presented in [113].

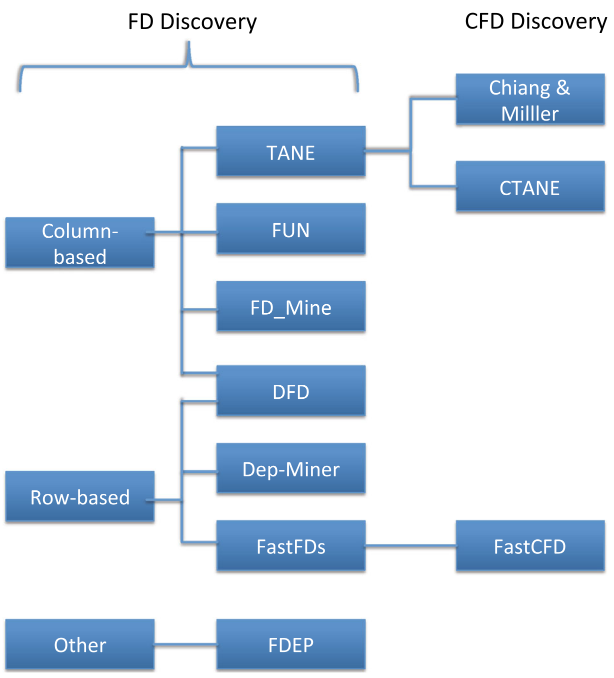

Fig. 3 Classification of algorithms for functional dependency discovery and their extensions to conditional functional dependencies

5.2.1 Column-based algorithms

As was the case with uniques, Apriori-like approaches can help prune the space of FDs that need to be examined, thereby optimizing the first two lines of the above straightforward algorithms. TANE [75], FUN [110], and FD_Mine [139] are three algorithms that follow this strategy, with FUN and FD_Mine introducing additional pruning rules beyond TANE's based on the properties of FDs. They start with sets of single columns in the LHS and work their way up the powerset lattice in a *level-wise* manner. Since only minimal FDs need to be returned, it is not necessary to test possible FDs whose LHS is a superset of an already found FD with the same RHS. For instance, in Table 4, once we find that *phone* $\rightarrow$ *age* holds, we do not need to consider {first,phone} $\rightarrow$ age, {last,phone} $\rightarrow$ age, etc.

Additional pruning rules may be formulated from Armstrong's axioms, i.e., we can prune from consideration those FDs that are logically implied by those we have found so far. For instance, if we find that $A \rightarrow B$ and $B \rightarrow A$, then we can prune all LHS column sets including B, because A

and B are equivalent [139]. Another pruning strategy is to ignore columns sets that have the same number of distinct values as their subsets [110]. Returning to Table 4, observe that phone $\rightarrow$ first does not hold. Since $|\pi_{\text{phone}}(r)| = |\pi_{\text{last,phone}}(r)| = |\pi_{\text{age,phone}}(r)| = |\pi_{\text{last,age,phone}}(r)|$, we know that adding last and/or age to the LHS cannot lead to a valid FD with first on the RHS. To determine these cardinalities the approaches use a so-called partition data structure, which is similar to the PLIs of Sect. 5.1.3.

5.2.2 Row-based algorithms

Row-based algorithms examine pairs of tuples to determine LHS candidates. Dep-Miner [95] and FastFDs [136] are two examples; the FDEP algorithm [52] is also row-based, but the way it ultimately finds FDs that hold is different.

The idea behind row-based algorithms is to compute the so-called difference sets for each pair of tuples, which are the columns on which the two tuples differ. Table 5 enumerates the difference sets in the data from Table 4. Next, we can find candidate LHS's from the difference sets as follows. Pick a candidate RHS, say, phone. The difference sets that include phone, with phone removed are as follows: {first,last,age}, {first,age}, {age}, {last} and {first,last}. This means that there exist pairs of tuples with different values of phone and also with different values of these five difference sets. Next, we find minimal subsets of columns that have a non-empty intersection with each of these difference sets. Such subsets are exactly the LHS's of minimal FDs with phone as the RHS: If two tuples have different values of phone, they are guaranteed to have different values of the columns in the above minimal subsets, and therefore, they do not cause FD violations. Here, there is only one such minimal subset, {last,age}, giving {last,age} $\rightarrow$ phone. If we repeat this process for each possible RHS, and compute minimal subsets corresponding to the LHS's, we obtain the set of minimal FDs. The main difference among row-based FD discovery algorithms is in how they find the minimal subsets.

A recent approach to FD discovery is DFD, which adapts the column-based and row-based pruning of the unique discovery approach DUCC to the problem of FD discovery [6].

Table 5 Difference sets computed from Table 4

Tuple ID pair	Difference set
(1,2)	first, last, age, phone
(1,3)	first, age, phone
(1,4)	age, phone
(2,3)	last, phone
(2,4)	first, last, phone
(3,4)	first

DFD decomposes the attribute lattice into $|R|$ lattices, considering each attribute as a possible RHS of an FD. For the remaining $|R| - 1$ attributes, DFD applies a random walk approach by pruning supersets of FD LHS's and subsets of non-FD LHS's.

DFD has been experimentally compared to TANE, which is a column-based approach, and FastFDs, which is row-based [6]. The experiments confirm that row-based approaches work well on high-dimensional tables with a relatively small number of tuples, while column-based approaches, such as TANE, perform better on low-dimensional tables with a large number of rows. DFD, which benefits from row-based and column-based pruning, performs significantly better than TANE and FastFDs, unless the table has very many tuples and very few columns or vice versa.

5.2.3 Partial and approximate functional dependencies

While FDs were meant for schema design and were enforced by the database management system, there are many instances in which a database may not satisfy some FDs exactly. For example, the application semantics may have changed over time and FD enforcement was disabled, or the database may have been created by integrating conflicting data sources. As a result, it is useful to discover *partial* or *soft* FDs, i.e., those which "almost hold," perhaps with a few exceptions.

A common definition of "almost holding" or "confidence" is the relative size of the largest subset of r on which a given FD holds exactly divided by $|r|$ [58,85]. For example, if we remove tuple 1 from Table 4, the FD $last \rightarrow phone$ holds exactly, and therefore, its confidence is $\frac{3}{4}$. The CORDS system for finding soft FDs uses a slightly different definition: The confidence of $X \rightarrow A$ is $\frac{|\pi_X(r)|}{|\pi_{XA}(r)|}$ [76]. Other definitions involve computing the number of tuples or tuple pairs that do not violate the FD divided by $|r|$ or $|r|^2$, respectively [85].

A related notion is that of *approximate* FD inference, in which partial or exact FDs are generated from a sample of a relation [76,85]. Of course, even if an FD holds exactly on a subset of a relation, it may hold partially on the whole relation. Approximate FD inference is appealing from a computational standpoint as it requires only a sample of the data.

5.2.4 Conditional functional dependencies

Conditional functional dependencies (CFDs), proposed in [46], encode FDs that hold only on well-defined subsets of r. For instance, {first,last} $\rightarrow$ age does not hold on the entire relation in Table 4, but it does hold on a subset of it where first = Eve. Formally, a CFD consists of two parts: an embedded FD $X \rightarrow A$ and an accompanying *pattern tuple* with attributes XA. Each cell of a pattern tuple contains a value from the corresponding attribute's domain or a wildcard symbol "_". A pattern tuple identifies a subset of a relation instance in a natural way: A tuple r_i matches a pattern tuple if it agrees on all of its non-wildcard attributes. In the above example, we can formulate a CFD with an embedded FD {first,last} $\rightarrow$ age and a pattern tuple (Eve, _, _), meaning that the embedded FD holds only on tuples which match the pattern, i.e., those with first = Eve. We define the *support* of a pattern tuple as the fraction of tuples in r that it matches; for example, the support of (Eve, _, _) in Table 4 is $\frac{2}{4}$.

An important special case occurs when the pattern tuple has no wildcards. For example, the following (admittedly accidental) CFD holds on Table 4: age $\rightarrow$ phone with a pattern tuple (32, 1234). In other words, if age = 32, then phone = 1234. These special cases, which resemble instance-level association rules (that have 100 % *confidence*), are referred to as *constant* CFDs.

Additionally, as was the case with traditional FDs, we can define approximate CFDs as those that hold on the subset specified by the pattern tableau with some exceptions. For the case of confidence defined as the minimum number of tuples that must be removed to make the CFD hold, [32] gives algorithms for computing summaries that allow the confidence of a CFD to be estimated with guaranteed accuracy.

CFD discovery involves a larger search space than FD discovery: In addition to detecting embedded FDs, we must also find the pattern tuples. CFD discovery algorithms typically extend existing FD discovery algorithms: For example, CTANE [47] and the algorithm from [24] extend TANE, while FastCFD [47] extends FastFDs (see Fig. 3).

Additionally, two simpler problems have been studied. The first is to discover pattern tuples given an embedded FD [59]. The output of this technique is an (approximately) smallest set of pattern tuples, each leading to an approximate CFD with a confidence exceeding a user-supplied confidence threshold, the union of which has a support that exceeds a user-supplied support threshold. The second problem is to report only the constant CFDs. For this problem, CFDMiner has been proposed CFDs [47], which is based on frequent itemset mining, as well as FACD [91], which includes more pruning rules. Also, CFUN, an extension of FUN to generating *frequent* constant CFDs that exceed a given support threshold, has been proposed in [42].

5.3 Inclusion dependencies

An *inclusion dependency* (IND) between column A of relation R and column B of relation S, written $R.A \subseteq S.B$, or $A \subseteq B$ when the relations are clear from context, asserts that each value of A appears in B. Similarly, for two sets of columns X and Y, we write $R.X \subseteq S.Y$, or $X \subseteq Y$, when each distinct combinations of values in X appears in Y. We refer to $R.A$ or $R.X$ as the left-hand side (LHS) and $S.B$ or $S.Y$ as the right-hand side (RHS). INDs with a single-column LHS and RHS are referred to as *unary* and those with multiple columns in the LHS and RHS are called *n-ary*.

A naive solution to IND discovery in relation instances r and s is to try to match each possible LHS with each possible RHS, as shown below.

```
For each column combination $X$ in $R$
  For each column combination $Y$ in $S$
  with $|Y| = |X|$
    If $\forall x \in \pi_X(r)\ \exists y \in \pi_Y(s)$ such that $x = y$
      Return $X \subseteq Y$
```

Note that for any considered X and Y, we can stop as soon as we find a value combination of X that does not appear in Y. Still, this is not an efficient approach as it repeatedly scans r and s when testing the possible LHS and RHS combinations.

5.3.1 Generating unary inclusion dependencies

For the special case of unary INDs, a common approach is to preprocess the data to speed up the subsequent IND discovery. De Marchi et al. [101] propose a technique that scans the database and builds value indices, which are similar to inverted indices. Table 6 shows excerpts of two relations instances, one with columns A and B and the other with columns C and D, and the corresponding value index. The index contains an entry for each value occurring in the database, followed by a list of columns in which this value appears. It is now straightforward to find the INDs: For each possible LHS column, we check if there exists another column that occurs in every row of the value index that contains the LHS column. In Table 6, we have $A \subseteq C$ (whenever A appears in the value index, so does B) and $D \subseteq B$.

The SPIDER algorithm [14] is another example, which preprocesses the data by sorting the values of each column and writing them to disk. Next, each sorted stream, corresponding to the values of one particular attribute, is consumed in parallel in a cursor-like manner, and an IND $A \subseteq B$ can be discarded as soon as we detect a value in A that is not present in B.

Table 6 Excerpts of two relation instances and the corresponding value index

A	B	C	D	Value	Columns
1	3	1	3	1	A, C
1	4	2	3	2	A, C
2	3	4	4	3	B, D
1	5	7	4	4	B, D
				5	B
				7	C

5.3.2 Generating n-ary inclusion dependencies

Once all unary INDs have been discovered, De Marchi et al. [101] give a level-wise algorithm, similar to the TANE algorithm for FD discovery, which constructs INDs with i columns from those with $i-1$ columns and prunes INDs that cannot be true. Additionally, hybrid algorithms have been proposed in [87,102] that combine bottom-up and top-down traversal for additional pruning.

The BINDER algorithm uses divide and conquer principles to handle larger datasets than related work [114]. In the divide step, it splits the input dataset horizontally into partitions and vertically into buckets with the goal to fit each partition into main memory. In the conquer step, BINDER then validates the set of all possible inclusion dependency candidates, which are created in the same fashion as in [101], against the partitions. Processing one partition after another, the validation constructs two indexes on each partition, a dense index and an inverted index, and uses them to efficiently prune invalid candidates from the result set.

5.3.3 Partial and approximate inclusion dependencies

Similar to partial FDs, partial INDs have been defined as those that almost hold. Using the notion of removing the fewest tuples so that the remainder satisfies the IND exactly, we can define the strength or confidence of a partial IND $X \subseteq Y$ as $\frac{|\pi_X(r)|-|\pi_X(r)/\pi_Y(r)|}{|\pi_X(r)|}$ [96,101]. That is, the confidence is the number of distinct values of X that appear in Y divided by the number of distinct values of X. An equivalent bag-semantics version of this definition is to divide the number of tuples whose X-values appear in Y by the total number of tuples [61]. According to both definitions, the confidence of $B \subseteq D$ in Table 6 is $\frac{3}{4}$. Most of the algorithms discussed above can be extended to discover partial INDs.

5.3.4 Conditional inclusion dependencies

Similar to CFDs, *conditional inclusion dependencies* (CINDs) represent INDs that hold only on well-defined subsets of relations [19]. A CIND consists of an embedded standard IND $R.X \subseteq S.Y$ and an accompanying pattern tuple with attributes $R.X_p$ and $S.Y_p$, where $X \cap X_p = \emptyset$ and $Y \cap Y_p = \emptyset$. A CIND specifies that for the subset of R that matches the X_p-values of the pattern tuple, all the X-values must appear in Y, and furthermore, the Y_p values of these tuples in S must match the Y_p-values of the pattern tuple.

For example, suppose a business maintains a Customers table, keyed by cid, and including a column class indicating the class of the customer (e.g., gold or silver). Furthermore, suppose a Services table maintains the services that customers subscribe to, including a service id (sid), a cid and the type of service (e.g., hardware or software). Let Services.cid $\subseteq$ Customers.cid be the embedded IND and let (Services.type $=$ `software`, Customers.class $=$ `gold`) be a pattern tuple. This CIND asserts that the customer ids in the Services table must be drawn from the customer ids in the Customers table, and moreover, gold customers can obtain only software services. On the other hand, a pattern tuple Services.type $=$ `software` implies that only the software services must have customer ids drawn from those in the Customers table (e.g., perhaps hardware services are provided to customers stored in a different table).

Given an embedded IND, the algorithm from [61], which also applies to CFDs, finds pattern tuples that lead to partial CINDs with a confidence satisfying a user-supplied threshold. Similarly, Bauckmann et al. [13] start with a set of approximate INDs and find pattern tuples to turn these into CINDs; however, in contrast to [61], they are not constrained to a single embedded IND. The authors present two algorithms: CINDERELLA, which is based on the Apriori algorithm for association rule mining and employs a breadth-first traversal of the powerset lattice, and PLI, which employs a depth-first traversal instead.

5.3.5 Generating foreign keys

IND discovery is a precursor to foreign key detection: A foreign key must satisfy the corresponding inclusion dependency but not all INDs are foreign keys. For example, multiple tables may contain auto-increment columns that serve as surrogate keys, and while inclusion dependencies among them may exist, they are not foreign keys. Once INDs have been discovered, additional heuristics have been proposed, which essentially rank the discovered INDs according to their likelihood of being foreign keys [96,123,143]. A very simple rule may be that if the LHS and RHS have similar names, then A may be a foreign key. It is also useful to examine the set of discovered INDs as a whole: For instance, foreign keys usually are not also primary keys that serve as foreign keys for other tables, and furthermore, a primary key is often referenced by multiple foreign keys in multiple tables, meaning that a primary key should appear in the RHS of multiple INDs, with the LHS's being the foreign keys. More complex rules may reference value distributions; for example, the values in a foreign key column should form a random sample of the values in the corresponding primary key column.

5.4 Other dependencies

Having outlined the algorithms for discovering traditional dependencies and their extensions, we now discuss other types of dependencies related to data profiling. Recently, an extension of FastFDs called FastDC was proposed for discovering *denial constraints*, which are universally quantified first-order logic formulas that subsume FDs, CFDs, INDs and many others [29].

Also, functional dependencies have recently been generalized to *differential dependencies* in [128]. A differential dependency $X \rightarrow Y$ states that if two tuples have "close" values of X (say, the edit distance between them is small), then their A values must also be close.[4] For example, in a financial database, it may be true that if two tuples have similar values of date (e.g., within seven days), then their price values must also be similar (e.g., within 100 dollars). Row- and column-based approaches to discovering differential dependencies were given in [128].

Another interesting class of dependencies involve *order*. For instance, it may be useful to discover that if r is sorted on some attribute A, it is also sorted on B, which gives an order dependency between A and B [56]. This concept was generalized in [57], which proposed *sequential dependencies* (SDs). An SD states that when sorted on A, any two consecutive values of B must be within a predefined range. Given a complete SD, including the attributes A and B as well as the range, [57] gives an algorithm for discovering ranges of values of A in which the SD is approximately satisfied. To the best of our knowledge, the general problem of SD discovery from data is open.

5.5 Summary and discussion

Dependency discovery has been a popular research area in data management. Many of the algorithms and techniques for dependency discovery are based upon classical data mining solutions, such as the Apriori algorithm for efficient generation of association rules. Additional technical challenges arise in the context of conditional dependencies, and novel search space pruning strategies have been developed based on the properties of the given dependencies.

Data profiling results can be not only complex, but also very large. For instance, it is not uncommon to find thousands of functional dependencies in a given dataset. To handle this and focus users on the most important, interesting, or surprising ones, ranking profiling results can help, as Chu et al. [29] show for denial constraints. They suggest two functions, namely succinctness and coverage, to assess their interestingness. Similar interestingness functions for CFDs are given by Chiang and Miller [24]. Additionally, Andritsos et al. [9] show how to rank FDs according to their information content. Furthermore, as we discussed earlier, post-processing methods have been proposed to determine which of the discovered inclusion dependencies are likely to be foreign keys; however, we are not aware of corresponding techniques for uniques and FDs.

6 Profiling tools

Whenever data are too voluminous to fit on a screen or a sheet of paper, data profiling is performed. Even lacking explicit profiling tools, much can already be done with data management tools, such as spreadsheet software, SQL queries, search capabilities of text editors or simply by "eyeballing" the data. Such methods to become acquainted with a new set of data are probably familiar to most readers. The simple method of *sorting* the values of a column can already reveal minimum and maximum values, and scrolling through that sorted data intuits the value distribution, including the number of null values, which are typically sorted to the very beginning or end, and the uniqueness of a column. Finding the median or average values requires additional calculations, whereas it is infeasible to detect dependencies with such simple means.

To allow a more powerful and integrated approach to data profiling, software companies have developed data profiling tools, mostly to profile data residing in relational databases. Most tools discussed in this survey are part of a larger software suite, either for data integration or for data cleansing. We first give an overview of tools that were created in the context of a research project (see Table 7 for a listing). Then, we give a brief glimpse of the vast set of commercial tools with profiling capabilities (see Table 8 for a listing).

6.1 Research tools

In the research literature, data profiling tools are often embedded in data cleaning systems. For example, the Bellman [38] data quality browser supports column analysis (counting the number of rows, distinct values, and NULL values, finding the most frequently occurring values, etc.), and key detection (up to four columns). It also provides a column similarity functionality that finds columns whose value or n-gram distributions are similar; this is helpful for discovering potential foreign keys and join paths. Furthermore, an interesting application of Bellman was to profile the evolution of a database using value distributions and correlations [37]: Which tables change over time and in what ways (insertions, deletions, modifications), and which groups of tables tend to change in the same way. The Potters Wheel tool [122] also supports column analysis, in particular, detecting data types and syntactic structures/patterns.

Data profiling functionality is also included in the MADLib toolkit for scalable in-database analytics [71], including column statistics, such as count, count distinct,

[4] Differential dependencies also generalize *matching dependencies* [49] (if two tuples have close values of X, their A values must be exactly the same) and *metric functional dependencies* [89] (if two tuples have the same values of X, their A values must be close).

Table 7 Research tools with data profiling capabilities

Tool	Main goal	Profiling capabilities
Bellman [38]	Data quality browser	Column statistics, column similarity, candidate key discovery
Potters Wheel [122]	Data quality, ETL	Column statistics (including value patterns)
Data Auditor [58]	Rule discovery	CFD and CIND discovery
RuleMiner [28]	Rule discovery	Denial constraint discovery
MADLib [71]	Machine learning	Simple column statistics

Table 8 Commercial data profiling tools/components with their primary capabilities and application areas

Vendor and product	Features → Focus
Attacama DQ Analyzer	Statistics, patterns, uniques → Data exploration, ETL
IBM InfoSphere Information Analyzer	Statistics, patterns, multi-column dependencies → Data exchange, integration, cleansing
Informatica Data Quality	Structure, completeness, anomalies, dependencies → Business rules, cleansing
Microsoft SQL Server Data Profiling Task	Statistics, patterns, dependencies → ETL, cleansing
Oracle Enterprise Data Quality	Statistics, patterns, multi-column dependencies, text profiling → Quality assessment, business rules, cleansing
Paxata Adaptive Data Preparation	Statistics, histograms, semantic data types → Exploration, cleansing, sharing
SAP Information Steward	Statistics, patterns, semantic data types, dependencies → ETL, modeling, cleansing
Splunk Enterprise / Hunk	Patterns, data mining → Search, analytics, visualization
Talend Data Profiler	Statistics, patterns, dependencies → ETL, cleansing
Trifacta	Statistics, patterns → Quality assessment, data transformation

minimum and maximum values, quantiles, and the *k* most frequently occurring values.

Recent data quality tools are dependency-driven: Classical dependencies, such as FDs and INDs, as well as their conditional extensions, may be used to express the intended data semantics, and dependency violations may indicate possible data quality problems. Most research systems require users to supply data quality rules and dependencies, such as GDR [138], Nadeef [34], Semandaq [45] and StreamClean [84]. These systems focus on languages for specifying rules and generating repairs. However, data quality rules are not always known Apriori in unfamiliar and undocumented datasets, in which case data profiling, and dependency discovery in particular, is an important prerequisite to data cleaning. Notably, many of these systems perform a focused profiling of counting the number of inconsistent tuples with respect to the given rules.

There are at least two research prototype systems that perform rule discovery to some degree: Data Auditor [58] and RuleMiner [28]. Data Auditor requires an FD as input and generates corresponding CFDs from the data. Additionally, Data Auditor considers FDs similar to the one that is provided by the user and generates corresponding CFDs. The idea is to see if a slightly modified FD can generate a more suitable CFD for the given relation instance. On the other hand, RuleMine does not require any rules as input and instead it is designed to generate all reasonable rules from a given dataset. RuleMiner expresses the discovered rules as *denial constraints*, which are universally quantified first-order logic formulas that subsume FDs, CFDs, INDs and many others. Some of the rules it finds are instance-specific and therefore more general than those a typical data profiling tool would find; for example, in a database of income tax records, RuleMiner might find that if one person, A, has a higher salary than another, B, then Person A must have a higher tax rate than Person B.

6.2 Commercial tools

Because data profiling is such an important capability for many data management tasks, there are various commercial data profiling applications. In many cases, they are a part of a data quality / data cleansing tool suite, to support the use-case of profiling for frequent patterns or rules and then cleaning those records that violate them. In addition, most Extract–Transform–Load tools have some profiling capabilities.

Table 8 mentions prominent examples of current commercial tools, together with their capabilities and application focus, based on the respective product documentations. It is beyond the scope of this survey to provide a market overview or compile feature matrices. We also deliberately refrain from providing static URLs for the various products, because commercial Web sites are too fickle.

Finally, and as mentioned before, every database management system collects and maintains base statistics about the tables it manages. However, they do not readily expose those metadata, the metadata are not always up-to-date and sometimes based only on samples, and their scope is usually limited to simple counts and cardinalities.

7 Next generation profiling

Recent trends in data management have added new challenges but also opportunities for data profiling. First, under the *big data* umbrella, industry and research have turned their attention to data that they do not own or have not made use of yet. Data profiling can help assess which data might be useful and reveals the yet unknown characteristics of such new data. Second, much of the data that shall be exploited is of non-traditional type for data profiling, i.e., non-relational, non-structured (textual), and heterogeneous. And it is often truly "big," both in terms of schema and in terms of data. Many existing profiling methods cannot adequately handle that kind of data: Either they do not scale well, or there simply are no methods yet. Third, different and new data management architectures and frameworks have emerged, including distributed systems, key-value stores, multi-core- or main-memory-based servers, column-oriented layouts, streaming input, etc. We discuss some of these trends and their implications toward data profiling. A more elaborate overview of upcoming challenges of data profiling is in [108].

7.1 Profiling for integration

An important use-case of traditional data profiling methods is data integration. Knowledge about the properties of different data sources is important to create correct schema mappings and data transformations, and to correctly standardize and cleanse the data. For instance, knowledge of inclusion dependencies might hint upon ways to join two yet unrelated tables.

However, data profiling can reach beyond such supportive tasks and 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 manmonths or in terms of which tools are needed.

Like integration projects themselves, integrability has two dimensions, namely schematic fit and data fit. *Schematic fit* is the degree to which two schemata complement and overlap each other and can be determined using schema matching techniques [44]. Smith et al. [127] have recognized that schema matching techniques often play the role of profiling tools: Rather than using them to derive schema mappings and perform data transformation, they might assess project feasibility. Finally, the mere matching of schema elements might not suffice as a profiling-for-integration result: Additional column metadata can provide further details about the integration difficulty.

Data fit is the (estimated) number of real-world objects that are represented in both datasets, or that are represented multiple times in a single dataset and how different they are. Such multiple representations are typically identified using entity matching methods (also known as record linkage, duplicate detection, etc.) [27]. However, estimating the number of matches without actually performing the matching on the entire dataset is an open problem.

7.2 Profiling non-relational data

With the rapid growth of the World Wide Web, semi-structured data, such as XML and RDF data, and non-structured data, such as text document corpora, have become more important. The more flexible structure of non-relational datasets opens new challenges for profiling algorithms. So far, most methods apply only to or were developed for relational data. Below, we give an overview of both existing work that applies traditional profiling algorithms, as well as existing work about data-model-specific profiling approaches, to non-relational data. We focus on the three most relevant non-relational data formats: XML, RDF, and text documents.

7.2.1 XML

XML is the quasi-standard for exchanging data on the Web. Many applications, especially Web services, provide their results as XML documents. Because the XML structure explicitly contains markup and schema information, different profiling approaches have to be considered. Apart from that, Web services themselves are accessible through XML documents, such as WSDL and SOAP files, which are also worth profiling for Web service inspection and categorization.

There has already been a number of research approaches and proposals with a focus on statistical analysis of XML-formatted data. They concentrate either on the DTD structure, the XSD schema structure, or the inherent structure of XML documents. The analysis concentrates on gathering statistics about the number of root elements, attributes, the depth of content models, etc. [26,105,106,124].

Further approaches focus on algorithms that identify traditional relational dependencies in XML data. While Vincent et al. extend the notion of FDs to XML data [132], Yu et al.s [140] present an approach for discovering redundancies based on identified XML FD. There have also been adaptations of unique and key discovery concepts and algorithms to XML data [22]. Due to the more relaxed structure of XML, these approaches identify approximate keys [62] or validate the consistency of the identified keys against XSD definitions [10].

As many XML documents do not refer to a specific schema, a relevant application of profiling approaches is to support the process of schema extraction [17,69]. Additionally, the vast amount of existing documents do not always comply to specified syntactical rules [88], which can be identified via appropriate profiling techniques.

7.2.2 RDF

Although profiling tasks for XML data can easily be adapted to RDF datasets and vice versa, the requirement for RDF data to be machine readable and its important use-case Linked Open Data (LOD) give rise to RDF-specific challenges for data profiling. There are already some tools that generate metadata for a given RDF dataset. For example, LODStats is a stream-based approach for gathering comprehensive statistics about RDF datasets [12].

ProLOD++ provides additional functionalities by applying clustering and rule mining techniques [1]. When profiling RDF data, there are many interesting metadata beyond simple statistics and patterns of RDF statement elements, including synonymously used properties [4], inverse relationships of properties, the conformity and consistence of RDF structured data to the corresponding ontology [2], and the distribution of literals and de-referenceable resource URIs from different namespaces.

Because of the heterogeneity of interlinked sources, it is vital to identify where specific facts come from and how reliable they are. Therefore, another interesting task for profiling RDF data is provenance analysis [18].

7.2.3 Text

Many text analysis approaches and applications can be regarded as text profiling tasks. Statistical methods are used for tasks, such as information extraction [125], part-of-speech tagging [20], and text categorization [83].

Specifically, in the field of author attribution, there has been research on defining interesting features, such as word-length distributions, average number of syllables [73].

Additionally, linguistic metrics, such as distinctiveness, type-token ratio, and Simpson's index have been proposed to measure the style and diversity of text documents. The task of profiling can target single documents, such as a paper or a book, as well as sets of documents, such as Web document corpora, product reviews, or user comments. More sophisticated applications that use metadata generated through profiling include sentiment analysis and opinion mining [93,112].

7.3 Profiling dynamic data

Data profiling describes an instance of a dataset at a particular time. Since many applications work on frequently changing data, it is desirable to re-profile a dataset after a change, such as a deletion, insertion, or update, in order to obtain up-to-date metadata. Simple aggregates are easy to maintain incrementally, but many statistics needed for column analysis, such as distinct value counts, cannot be maintained exactly using limited space and time. For these aggregates, stream sketching techniques [53] may be used to maintain approximate answers. There are also techniques for continuously updating discovered association rules [131] and clusters [43].

Dependency detection may be too time-consuming for repeated execution on the entire dataset. Thus, it is necessary to incrementally update the metadata without processing the complete dataset again. One example is SWAN, an approach for unique discovery on dynamic datasets with insertions and deletions [5] as reported in Sect. 5.1.4. Also, Wang et al. present an approach for maintaining discovered FDs after data deletions [134]. From a data cleaning standpoint, there are solutions for incremental detection of FD and CFD violations [50], and incremental data repairing with respect to FDs and CFDs [30]. In general, incremental solutions for FDs, CFDs, INDs, and CINDs on growing and changing datasets remain challenges for future research.

7.4 Profiling on new architectures

There are at least two database architecture trends that affect profiling. The first is column versus row storage. Column-store systems appear to have a natural computational advantage, at least in terms of the column analysis tasks we discussed in Sect. 3, since they can directly fetch the column of interest and compute statistics on it. However, if all columns are to be profiled, the entire dataset must be read and the only remaining advantage of column stores may be their potential compression. The second trend is that of distributed and cloud data management. This introduces additional profiling challenges, such as combining statistics from multiple nodes into final per-column analysis. There has been some work on detecting FD and CFD violations in a distributed database [48,50], but many other problems in this space, such as efficient dependency detection in distributed data, remain open.

7.5 Visualization

Because data profiling mostly targets human users, effectively visualizing any profiling results is of utmost importance. Only then can users interpret results and react to them. A suggestion for a visual data profiling tool is the Profiler system by Kandel et al. [81]. A strong cooperation between the database community, which produces the data and metadata to be visualized, and the visualization community, which enables users to understand and make use of the data, is needed.

8 Summary

In this article, we provided a comprehensive survey of the state of the art in data profiling: the set of activities and processes to determine metadata about a given database. We discussed single-column profiling tasks such as identifying data types, value distributions and patterns, and multi-column tasks such as detecting various kinds of dependencies. As the amount of data and users who require access to data increase, efficient and effective data profiling will continue to be an important data management problem in research and practice. While many data profiling algorithms have been proposed and implemented in research prototypes and commercial tools, further work is needed, especially in the context of profiling new types of data, supporting and leveraging new data management architectures, and interpreting and visualizing data profiling results.

References

1. Abedjan, Z., Grütze, T., Jentzsch, A., Naumann, F.: Mining and profiling RDF data with ProLOD++. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 1198–1201 (2014). Demo
2. Abedjan, Z., Lorey, J., Naumann, F.: Reconciling ontologies and the web of data. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 1532–1536 (2012)
3. Abedjan, Z., Naumann, F.: Advancing the discovery of unique column combinations. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 1565–1570 (2011)
4. Abedjan, Z., Naumann, F.: Synonym analysis for predicate expansion. In: Proceedings of the Extended Semantic Web Conference (ESWC), pp. 140–154 (2013)
5. Abedjan, Z., Quiané-Ruiz, J.-A., Naumann, F.: Detecting unique column combinations on dynamic data. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 1036–1047 (2014)
6. Abedjan, Z., Schulze, P., Naumann, F.: DFD: efficient functional dependency discovery. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 949–958 (2014)
7. Agrawal, D., Bernstein, P., Bertino, E., Davidson, S., Dayal, U., Franklin, M., Gehrke, J., Haas, L., Halevy, A., Han, J., Jagadish, H.V., Labrinidis, A., Madden, S., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Ross, K., Shahabi, C., Suciu, D., Vaithyanathan, S., Widom, J.: Challenges and opportunities with Big Data. Technical report, Computing Community Consortium. http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf (2012)
8. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 487–499 (1994)
9. Andritsos, P., Miller, R.J., Tsaparas, P.: Information-theoretic tools for mining database structure from large data sets. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 731–742 (2004)
10. Arenas, M., Daenen, J., Neven, F., Ugarte, M., Van den Bussche, J., Vansummeren, S.: Discovering XSD keys from XML data. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 61–72 (2013)
11. Astrahan, M.M., Schkolnick, M., Kyu-Young, W.: Approximating the number of unique values of an attribute without sorting. Inf. Syst. **12**(1), 11–15 (1987)
12. Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats—an extensible framework for high-performance dataset analytics. In: Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW), pp. 353–362 (2012)
13. Bauckmann, J., Abedjan, Z., Müller, H., Leser, U., Naumann, F.: Discovering conditional inclusion dependencies. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 2094–2098 (2012)
14. Bauckmann, J., Leser, U., Naumann, F., Tietz, V.: Efficiently detecting inclusion dependencies. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 1448–1450 (2007)
15. Benford, F.: The law of anomalous numbers. Proc. Am. Philos. Soc. **78**(4), 551–572 (1938)
16. Berti-Equille, L., Dasu, T., Srivastava, D.: Discovery of complex glitch patterns: a novel approach to quantitative data cleaning. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 733–744 (2011)
17. Bex, G.J., Neven, F., Vansummeren, S.: Inferring XML schema definitions from XML data. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 998–1009 (2007)
18. Böhm, C., Lorey, J., Naumann, F.: Creating void descriptions for web-scale data. J. Web Semant. **9**(3), 339–345 (2011)
19. Bravo, L., Fan, W., Ma, S.: Extending dependencies with conditions. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 243–254 (2007)
20. Brill, E.: Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. Comput. Linguist. **21**(4), 543–565 (1995)
21. Brin, S., Motwani, R., Silverstein, C.: Beyond market baskets: generalizing association rules to correlations. SIGMOD Rec. **26**(2), 265–276 (1997)
22. Buneman, P., Davidson, S.B., Fan, W., Hara, C.S., Tan, W.C.: Reasoning about keys for XML. Inf. Syst. **28**(8), 1037–1063 (2003)
23. Chandola, V., Kumar, V.: Summarization—compressing data into an informative representation. Knowl. Inf. Syst. **12**(3), 355–378 (2007)
24. Chiang, F., Miller, R.J.: Discovering data quality rules. Proc. VLDB Endow. **1**, 1166–1177 (2008)
25. Chiang, R.H.L., Cecil, C.E.H., Lim, E.-P.: Linear correlation discovery in databases: a data mining approach. Data Knowl. Eng. **53**(3), 311–337 (2005)

26. Choi, B.: What are real DTDs like? In: Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB), pp. 43–48 (2002)
27. Christen, P.: Data Matching. Springer, Berlin (2012)
28. Chu, X., Ilyas, I., Papotti, P., Ye, Y.: RuleMiner: data quality rules discovery. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 1222–1225 (2014)
29. Chu, X., Ilyas, I.F., Papotti, P.: Discovering denial constraints. Proc. VLDB Endow. **6**(13), 1498–1509 (2013)
30. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: consistency and accuracy. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 315–326 (2007)
31. Cormode, G., Garofalakis, M., Haas, P.J., Jermaine, C.: Synopses for massive data: samples, histograms, wavelets, sketches. Found. Trends Databases **4**(13), 1–294 (2011)
32. Cormode, G., Golab, L., Flip, K., McGregor, A., Srivastava, D., Zhang, X.: Estimating the confidence of conditional functional dependencies. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 469–482 (2009)
33. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In: Proceedings of the Symposium on Principles of Database Systems (PODS), pp. 263–272 (2006)
34. Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I.F., Ouzzani, M., Tang, N.: NADEEF: a commodity data cleaning system. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 541–552 (2013)
35. Das, A., Ng, W.-K., Woon, Y.-K.: Rapid association rule mining. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 474–481 (2001)
36. Dasu, T., Johnson, T.: Hunting of the snark: finding data glitches using data mining methods. In: Proceedings of the International Conference on Information Quality (IQ), pp. 89–98 (1999)
37. Dasu, T., Johnson, T., Marathe, A.: Database exploration using database dynamics. IEEE Data Eng. Bull. **29**(2), 43–59 (2006)
38. Dasu, T., Johnson, T., Muthukrishnan, S., Shkapenyuk, V.: Mining database structure; or, how to build a data quality browser. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 240–251 (2002)
39. Dasu, T., Loh, J.M.: Statistical distortion: consequences of data cleaning. Proc. VLDB Endow. **5**(11), 1674–1683 (2012)
40. Dasu, T., Loh, J.M., Srivastava, D.: Empirical glitch explanations. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 572–581 (2014)
41. Deshpande, A., Garofalakis, M., Rastogi, R.: Independence is good: dependency-based histogram synopses for high-dimensional data. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 199–210 (2001)
42. Diallo, T., Novelli, N., Petit, J.-M.: Discovering (frequent) constant conditional functional dependencies. Int. J. Data Min. Model. Manag. **4**(3), 205–223 (2012)
43. Ester, M., Kriegel, H.-P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in a data warehousing environment. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 323–333 (1998)
44. Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Berlin (2013)
45. Fan, W., Geerts, F., Jia, X.: Semandaq: a data quality system based on conditional functional dependencies. Proc. VLDB Endow. **1**(2), 1460–1463 (2008)
46. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. ACM Trans. Database Syst. **33**(2), 1–48 (2008)
47. Fan, W., Geerts, F., Li, J., Xiong, M.: Discovering conditional functional dependencies. IEEE Trans. Knowl. Data Eng. **23**(4), 683–698 (2011)
48. Fan, W., Geerts, F., Ma, S., Müller, H.: Detecting inconsistencies in distributed data. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 64–75 (2010)
49. Fan, W., Jia, X., Li, J., Ma, S.: Reasoning about record matching rules. Proc. VLDB Endow. **2**(1), 407–418 (2009)
50. Fan, W., Li, J., Tang, N., Yu, W.: Incremental detection of inconsistencies in distributed data. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 318–329 (2012)
51. Fernau, H.: Algorithms for learning regular expressions from positive data. Inf. Comput. **207**(4), 521–541 (2009)
52. Flach, P.A., Savnik, I.: Database dependency discovery: a machine learning approach. AI Commun. **12**(3), 139–160 (1999)
53. Ganguly, S.: Counting distinct items over update streams. Theor. Comput. Sci. **378**(3), 211–222 (2007)
54. Garofalakis, M., Keren, D., Samoladas, V.: Sketch-based geometric monitoring of distributed stream queries. Proc. VLDB Endow. **6**(10), 937–948 (2013)
55. Giannella, C., Wyss, C.: Finding minimal keys in a relation instance (1999). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.41.7086
56. Ginsburg, S., Hull, R.: Order dependency in the relational model. Theor. Comput. Sci. **26**, 149–195 (1983)
57. Golab, L., Karloff, H., Korn, F., Saha, A., Srivastava, D.: Sequential dependencies. Proc. VLDB Endow. **2**(1), 574–585 (2009)
58. Golab, L., Karloff, H., Korn, F., Srivastava, D.: Data auditor: exploring data quality and semantics using pattern tableaux. Proc. VLDB Endow. **3**(1–2), 1641–1644 (2010)
59. Golab, L., Karloff, H., Korn, F., Srivastava, D., Bei, Y.: On generating near-optimal tableaux for conditional functional dependencies. Proc. VLDB Endow. **1**(1), 376–390 (2008)
60. Golab, L., Korn, F., Srivastava, D.: Discovering pattern tableaux for data quality analysis: a case study. In: Proceedings of the International Workshop on Quality in Databases (QDB), pp. 47–53 (2011)
61. Golab, L., Korn, F., Srivastava, D.: Efficient and effective analysis of data quality using pattern tableaux. IEEE Data Eng. Bull. **34**(3), 26–33 (2011)
62. Grahne, G., Zhu, J.: Discovering approximate keys in XML data. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 453–460 (2002)
63. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H.: Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub totals. Data Min. Knowl. Discov. **1**(1), 29–53 (1997)
64. Gunopulos, D., Khardon, R., Mannila, H., Sharma, R.S.: Discovering all most specific sentences. ACM Trans. Database Syst. **28**, 140–174 (2003)
65. Haas, P.J., Naughton, J.F., Seshadri, S., Stokes, L.: Sampling-based estimation of the number of distinct values of an attribute. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 311–322 (1995)
66. Hainaut, J.-L., Henrard, J., Englebert, V., Roland, D., Hick, J.-M.: Database reverse engineering. In: Liu, L., Tamer Özsu, M. (eds.) Encyclopedia of Database Systems, pp. 723–728. Springer, Heidelberg (2009)
67. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2), 1–12 (2000)
68. Hanrahan, P.: Analytic database technology for a new kind of user—the data enthusiast (keynote). In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 577–578 (2012)
69. Hegewald, J., Naumann, F., Weis, M.: XStruct: efficient schema extraction from multiple and large XML databases. In: Proceed-

ings of the International Workshop on Database Interoperability (InterDB) (2006)
70. Heise, A., Quiané-Ruiz, J.-A., Abedjan, Z., Jentzsch, A., Naumann, F.: Scalable discovery of unique column combinations. Proc. VLDB Endow. **7**(4), 301–312 (2013)
71. Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., Kumar, A.: The MADlib analytics library or MAD skills, the SQL. Proc. VLDB Endow. **5**(12), 1700–1711 (2012)
72. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining—a general survey and comparison. SIGKDD Explor. **2**(1), 58–64 (2000)
73. Holmes, D.I.: Authorship attribution. Comput. Humanit. **28**, 87–106 (1994)
74. Hua, M., Pei, J.: Cleaning disguised missing data: a heuristic approach. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 950–958 (2007)
75. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2), 100–111 (1999)
76. Ilyas, I.F., Markl, V., Haas, P.J., Brown, P., Aboulnaga, A.: CORDS: automatic discovery of correlations and soft functional dependencies. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 647–658 (2004)
77. Ioannidis, Y.: The history of histograms (abridged). In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 19–30 (2003)
78. Jain, A.K., Narasimha Murty, M., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999)
79. Johnson, T.: Encyclopedia of Database Systems, chapter Data Profiling. Springer, Heidelberg (2009)
80. Kache, H., Han, W.-S., Markl, V., Raman, V., Ewen, S.: POP/FED: progressive query optimization for federated queries in DB2. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 1175–1178 (2006)
81. Kandel, S., Parikh, R., Paepcke, A., Hellerstein, J., Heer, J.: Profiler: integrated statistical analysis and visualization for data quality assessment. In: Proceedings of Advanced Visual Interfaces (AVI), pp. 547–554 (2012)
82. Kang, J., Naughton, J.F.: On schema matching with opaque column names and data values. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 205–216 (2003)
83. Keim, D.A., Oelke, D.: Literature fingerprinting: a new method for visual literary analysis. In: Proceedings of Visual Analytics Science and Technology (VAST), pp. 115–122 (2007)
84. Khoussainova, N., Balazinska, M., Suciu, D.: Towards correcting input data errors probabilistically using integrity constraints. In: Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), pp. 43–50 (2006)
85. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. In: Proceedings of the International Conference on Database Theory (ICDT), pp. 129–149 (1995)
86. Koehler, H., Leck, U., Link, S., Prade, H.: Logical foundations of possibilistic keys. In: Fermé, E., Leite, J. (eds.) Logics in Artificial Intelligence, volume 8761 of Lecture Notes in Computer Science, pp. 181–195. Springer, Heidelberg (2014)
87. Koeller, A., Rundensteiner, E.A.: Heuristic strategies for the discovery of inclusion dependencies and other patterns. J. Data Semant. V. **3870**, 185–210 (2006)
88. Korn, F., Saha, B., Srivastava, D., Ying, S.: On repairing structural problems in semi-structured data. Proc. VLDB Endow. **6**(9), 601–612 (2013)
89. Koudas, N., Saha, A., Srivastava, D., Venkatasubramanian, S.: Metric functional dependencies. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 1275–1278 (2009)
90. Laney, D.: 3D data management: controlling data volume, velocity and variety. Technical report, Gartner (2001)
91. Li, J., Liu, J., Toivonen, H., Yong, J.: Effective pruning for the discovery of conditional functional dependencies. Comput. J. **56**(3), 378–392 (2013)
92. Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Jagadish, H.V.: Regular expression learning for information extraction. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 21–30 (2008)
93. Liu, B.: Sentiment analysis and subjectivity. Handbook of Natural Language Processing, 2nd edn. Chapman and Hall/CRC, London (2010)
94. Liu, J., Li, J., Liu, C., Chen, Y.: Discover dependencies from data—a review. IEEE Trans. Knowl. Data Eng. **24**(2), 251–264 (2012)
95. Lopes, S., Petit, J.-M., Lakhal, L.: Efficient discovery of functional dependencies and Armstrong relations. In: Proceedings of the International Conference on Extending Database Technology (EDBT), pp. 350–364 (2000)
96. Lopes, S., Petit, J.-M., Toumani, F.: Discovering interesting inclusion dependencies: application to logical database tuning. Inf. Syst. **27**(1), 1–19 (2002)
97. Lucchesi, C.L., Osborn, S.L.: Candidate keys for relations. J. Comput. Syst. Sci. **17**(2), 270–279 (1978)
98. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with Cupid. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 49–58 (2001)
99. Mannino, M.V., Chu, P., Sager, T.: Statistical profile estimation in database systems. ACM Comput. Surv. **20**(3), 191–221 (1988)
100. De Marchi, F., Lopes, S., Petit, J.-M.: Efficient algorithms for mining inclusion dependencies. In: Proceedings of the International Conference on Extending Database Technology (EDBT), pp. 464–476 (2002)
101. De Marchi, F., Lopes, S., Petit, J.-M.: Unary and n-ary inclusion dependency discovery in relational databases. J. Intell. Inf. Syst. **32**, 53–73 (2009)
102. De Marchi, F. , Petit, J.-M.: Zigzag: a new algorithm for mining large inclusion dependencies in databases. In: Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 27–34 (2003)
103. Markowitz, V.M., Makowsky, J.A.: Identifying extended entity-relationship object structures in relational schemas. IEEE Trans. Softw. Eng. **16**(8), 777–790 (1990)
104. Maydanchik, A.: Data Quality Assessment. Technics Publications, New Jersey (2007)
105. Mignet, L., Barbosa, D., Veltri, P.: The XML web: a first study. In: Proceedings of the International World Wide Web Conference (WWW), pp. 500–510 (2003)
106. Mlynkova, I., Toman, K., Pokorný, J.: Statistical analysis of real XML data collections. In: Proceedings of the International Conference on Management of Data (COMAD), pp. 15–26 (2006)
107. Morton, K., Balazinska, M., Grossman, D., Mackinlay, J.: Support the data enthusiast: challenges for next-generation data-analysis systems. Proc. VLDB Endow. **7**(6), 453–456 (2014)
108. Naumann, F.: Data profiling revisited. SIGMOD Rec. **42**(4), 40–49 (2013)
109. Naumann, F., Ho, C.-T., Tian, X., Haas, L., Megiddo, N.: Attribute classification using feature analysis. In: Proceedings of the International Conference on Data Engineering (ICDE), p 271 (2002)
110. Novelli, N., Cicchetti, R.: FUN: an efficient algorithm for mining functional and embedded dependencies. In: Proceedings of the

International Conference on Database Theory (ICDT), pp. 189–203 (2001)
111. Ntarmos, N., Triantafillou, P., Weikum, G.: Distributed hash sketches: scalable, efficient, and accurate cardinality estimation for distributed multisets. ACM Trans. Comput. Syst. **27**(1), 1–53 (2009)
112. Pang, B., Lee, L.: Opinion mining and sentiment analysis. Found. Trends Inf. Retr. **2**(1–2), 1–135 (2008)
113. Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.-P., Schönberg, M., Zwiener, J., Naumann, F.: Functional dependency discovery: an experimental evaluation of seven algorithms. Proc. VLDB Endow. **8**(10) (2015)
114. Papenbrock, T., Kruse, S., Quiané-Ruiz, J.-A., Naumann, F.: Divide & conquer-based inclusion dependency discovery. Proc. VLDB Endow. **8**(7), 774–785 (2015)
115. Park, J.S., Chen, M.-S., Yu, P.S.: Using a hash-based method with transaction trimming for mining association rules. IEEE Trans. Knowl. Data Eng. **9**, 813–825 (1997)
116. Petit, J.-M., Kouloumdjian, J., Boulicaut, J.-F., Toumani, F.: Using queries to improve database reverse engineering. In: Proceedings of the International Conference on Conceptual Modeling (ER), pp. 369–386 (1994)
117. Pipino, L., Lee, Y., Wang, R.: Data quality assessment. Commun. ACM **4**, 211–218 (2002)
118. Poosala, V., Haas, P.J., Ioannidis, Y.E., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 294–305 (1996)
119. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 486–495 (1997)
120. Pyle, D.: Data Preparation for Data Mining. Morgan Kaufmann, Burlington (1999)
121. Rahm, E., Do, H.-H.: Data cleaning: problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
122. Raman, V., Hellerstein, J.M.: Potters wheel: an interactive data cleaning system. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 381–390 (2001)
123. Rostin, A., Albrecht, O., Bauckmann, J., Naumann, F., Leser, U.: A machine learning approach to foreign key discovery. In: Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB) (2009)
124. Sahuguet, A., Azavant, F.: Building light-weight wrappers for legacy Web data-sources using W4F. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 738–741 (1999)
125. Sarawagi, S.: Information extraction. Found. Trends Databases **1**(3), 261–377 (2008)
126. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: GORDIAN: efficient and scalable discovery of composite keys. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 691–702 (2006)
127. Smith, K.P., Morse, M., Mork, P., Li, M.H., Rosenthal, A., Allen, M.D., Seligman, L.: The role of schema matching in large enterprises. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR) (2009)
128. Song, S., Chen, L.: Differential dependencies: reasoning and discovery. ACM Trans. Database Syst. **36**(3), 16:1–16:41 (2011)
129. Stonebraker, M., Bruckner, D., Ilyas, I.F., Beskales, G., Cherniack, M., Zdonik, S., Pagan, A., Xu, S.: Data curation at scale: the Data Tamer system. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR) (2013)
130. Chen, M., Hun, J., Yu, P.S.: Data mining: an overview from a database perspective. IEEE Trans. Knowl. Data Eng. **8**, 866–883 (1996)
131. Tsai, P.S.M., Lee, C.-C., Chen, A.L.P.: An efficient approach for incremental association rule mining. Methodologies for Knowledge Discovery and Data Mining. volume 1574 of Lecture Notes in Computer Science, pp. 74–83. Springer, Heidelberg (1999)
132. Vincent, M.W., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. ACM Trans. Database Syst. **29**(3), 445–462 (2004)
133. Vogel, T., Naumann, F.: Instance-based "one-to-some" assignment of similarity measures to attributes. In: Proceedings of the International Conference on Cooperative Information Systems (CoopIS), pp. 412–420 (2011)
134. Wang, S.-L., Tsou, W.-C., Lin, J.-H., Hong, T.-P.: Maintenance of discovered functional dependencies: incremental deletion. Intelligent Systems Design and Applications, volume 23 of Advances in Soft Computing, pp. 579–588. Springer, Heidelberg (2003)
135. Xindong, W., Zhang, C., Zhang, S.: Efficient mining of both positive and negative association rules. ACM Trans. Inf. Syst. **22**(3), 381–405 (2004)
136. Wyss, C., Giannella, C., Robertson, E.L.: FastFDs: a heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In: Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK), pp. 101–110 (2001)
137. Xu, R., Wunsch II, D.C.: Survey of clustering algorithms. IEEE Trans. Neural Netw. **16**(3), 645–678 (2005)
138. Yakout, M., Elmagarmid, A.K., Neville, J., Ouzzani, M.: GDR: a system for guided data repair. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 1223–1226 (2010)
139. Yao, H., Hamilton, H.J.: Mining functional dependencies from data. Data Min. Knowl. Discov. **16**(2), 197–219 (2008)
140. Yu, C., Jagadish, H.V.: Efficient discovery of XML data redundancies. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 103–114 (2006)
141. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
142. Zhang, M., Chakrabarti, K.: InfoGather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 145–156 (2013)
143. Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D.: On multi-column foreign key discovery. Proc. VLDB Endow. **3**(1–2), 805–814 (2010)
144. Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D.: Automatic discovery of attributes in relational databases. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp. 109–120 (2011)