Data Profiling with Metanome

Thorsten Papenbrock◆ Tanja Bergmann⋆ Moritz Finke⋆
Jakob Zwiener⋆ Felix Naumann◆

Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
⋆ firstname.lastname@student.hpi.de ◆firstname.lastname@hpi.de

ABSTRACT

Data profiling is the discipline of discovering metadata about given datasets. The metadata itself serve a variety of use cases, such as data integration, data cleansing, or query optimization. Due to the importance of data profiling in practice, many tools have emerged that support data scientists and IT professionals in this task. These tools provide good support for profiling statistics that are easy to compute, but they are usually lacking automatic and efficient discovery of complex statistics, such as inclusion dependencies, unique column combinations, or functional dependencies.

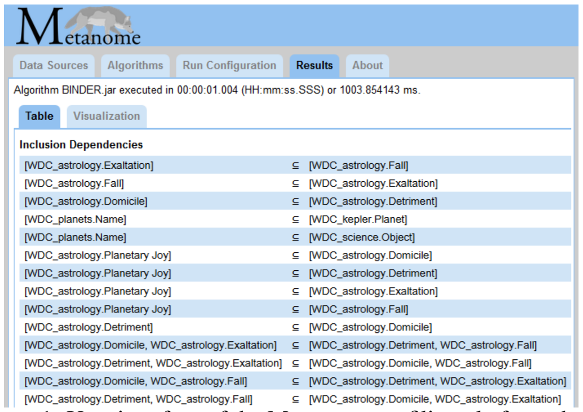
We present Metanome, an extensible profiling platform that incorporates many state-of-the-art profiling algorithms. While Metanome is able to calculate simple profiling statistics in relational data, its focus lies on the automatic *discovery of complex metadata*. Metanome's goal is to provide novel profiling algorithms from research, perform comparative evaluations, and to support developers in building and testing new algorithms. In addition, Metanome is able to rank profiling results according to various metrics and to visualize the, at times, large metadata sets.

1. DATA PROFILING

Data scientists and IT professionals are often confronted with datasets about which only little is known. To work with these datasets and to gain information from them, we must analyze their records for metadata: At first, we usually inspect the schema for its number of columns and the column labels. Then, we count the number of rows and measure the size of the data in terms of megabyte. We also take manageable samples of records to get an impression on the dataset's values and to understand what the data is about. All such profiling can be done with a standard text editor.

More complex tasks, however, require more demanding metadata: If we, for instance, aim to load a dataset from a file into a database, we require the columns' data types. To optimize aggregation queries, we need statistical metadata, such as the minimum, maximum, average, and median of all columns. We might, furthermore, want to compress or re-encode the data and must, hence, check the columns for their values' entropy, lengths, and filling degree. Computing all these metadata requires special profiling

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.



Algorithm BINDER.jar executed in 00:00:01.004 (HH:mm:ss.SSS) or 1003.854143 ms.

Inclusion Dependencies

[WDC_astrology.Exaltation]	⊆	[WDC_astrology.Fall]
[WDC_astrology.Fall]	⊆	[WDC_astrology.Exaltation]
[WDC_astrology.Domicile]	⊆	[WDC_astrology.Detriment]
[WDC_planets.Name]	⊆	[WDC_kepler.Planet]
[WDC_planets.Name]	⊆	[WDC_science.Object]
[WDC_astrology.Planetary Joy]	⊆	[WDC_astrology.Domicile]
[WDC_astrology.Planetary Joy]	⊆	[WDC_astrology.Detriment]
[WDC_astrology.Planetary Joy]	⊆	[WDC_astrology.Exaltation]
[WDC_astrology.Planetary Joy]	⊆	[WDC_astrology.Fall]
[WDC_astrology.Detriment]	⊆	[WDC_astrology.Domicile]
[WDC_astrology.Domicile, WDC_astrology.Exaltation]	⊆	[WDC_astrology.Detriment, WDC_astrology.Fall]
[WDC_astrology.Detriment, WDC_astrology.Exaltation]	⊆	[WDC_astrology.Domicile, WDC_astrology.Fall]
[WDC_astrology.Domicile, WDC_astrology.Fall]	⊆	[WDC_astrology.Detriment, WDC_astrology.Exaltation]
[WDC_astrology.Detriment, WDC_astrology.Fall]	⊆	[WDC_astrology.Domicile, WDC_astrology.Exaltation]

Figure 1: User interface of the Metanome profiling platform showing a result set of inclusion dependencies.

methods, but still the computation needs only a single pass over all records and is, hence, easy to achieve. This is why most current data profiling tools provide good support for such metadata.

In practice, however, many data management tasks require metadata that is much harder to discover: To define key columns, for instance, we need to find *unique column combinations* that serve as key candidates. If a dataset comprises multiple relations, we might want to discover foreign keys and, hence, *inclusion dependencies* as their prerequisite. In order to efficiently store the data, we perform schema normalization that builds upon the discovery of *functional dependencies*. An IT professional can, furthermore, optimize the sortation of records in a dataset using *order dependencies*. The list of important metadata continues, but tool support for their discovery is sparse, as research is still ongoing in these areas.

For this reason, we present with Metanome[1] an open profiling platform that integrates many state-of-the-art metadata discovery algorithms. Metanome serves two important use cases: On the one hand, the Metanome tool is built for database administrators and IT professionals that require efficient metadata discovery algorithms in any kind of data profiling project; on the other hand, the Metanome framework is designed to support developers and researchers in building, testing, and publishing new algorithms. In both cases, the interpretation of profiling results plays a major role. Therefore, Metanome provides several result management techniques, such as metadata ranking and visualization. Figure 1 shows Metanome's user interface and its basic result listing.

In this demo paper, we first introduce the Metanome platform by explaining its architecture, the modularization of algorithms, the user interaction, and the result handling (Section 2). Then, we show

[1]www.metanome.de

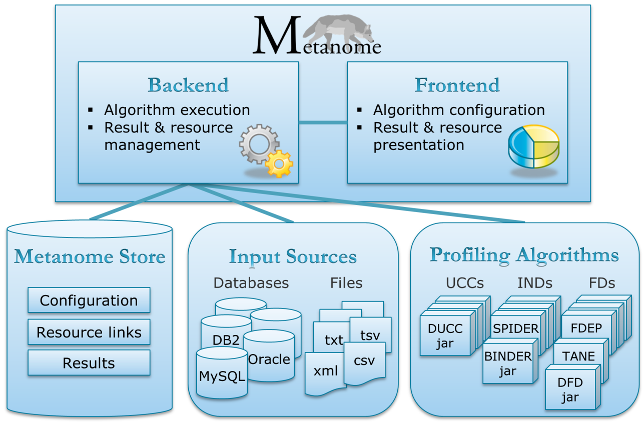


Figure 2: Architecture of the Metanome profiling platform.

how data profiling with Metanome works from the user perspective and we describe how Metanome supports the development of new algorithms (Section 3). After the description of our profiling system, we discuss what our demonstration will offer to attendees (Section 4).

2. THE DATA PROFILING PLATFORM

In this section, we discuss the architecture of Metanome. The Metanome project is an open source project available on GitHub[2]. A nightly build and more detailed documentation for developers can be found on the same page. The development of Metanome has the following design goals:

Simplicity. Metanome should be easy to setup and use, i.e., a user should be able to simply download and start the tool, add data, and begin profiling it.

Extensibility. New algorithms and datasets should be easily addable to the system without needing to change the system itself.

Standardization. All common tasks, such as tooling, input parsing, or result handling, should be integrated into the Metanome framework allowing the algorithms to focus on their specific solution strategies.

Flexibility. Metanome should make as few restrictions to the algorithms as possible in order to enable all algorithmic ideas.

In the following, we describe how these goals influenced Metanome's architecture, the framework's tasks, and the modularization of profiling algorithms.

2.1 Architecture

Metanome is a web-application that builds upon a classical three-tier architecture. Figure 2 illustrates this architecture: The data tier comprises the *Metanome store*, the *input sources*, and the *profiling algorithms*; the logic tier appears as *backend* component; and the presentation tier is represented by the *frontend* component. The division into server (data and logic tier) and client (presentation tier) components is an important design decision for a profiling platform, because data profiling is usually done from an IT professional's workstation on a remote server that holds the data and the necessary compute resources. Hence, the client can steer the profiling and analysis processes while the server performs the expensive profiling. For small profiling tasks, however, Metanome can also run as a desktop application. We now discuss the three tiers in more detail:

[2]`https://github.com/HPI-Information-Systems/Metanome`

Data tier. The *Metanome store* is a light-weight HSQLDB that stores operational metadata for the tool, such as configuration parameters, links to external resources, and statistics about previous profiling runs. The database is shipped with the Metanome tool and does not need to be installed separately. The data tier further comprises the *input sources*, which can be files or databases, and the *profiling algorithms*, which are precompiled jar-files. Both sources and algorithms are managed dynamically, meaning that they can be added or removed at runtime.

Logic tier. The *backend* executes the algorithms and manages the results. It provides methods for several common tasks to the algorithms. Input parsing, output processing, and algorithm parametrization are, in this way, standardized. This makes the profiling algorithms easier to develop, evaluate, and compare.

Presentation tier. The *frontend* provides a graphical web-interface to the user. This interface allows the user to add/remove input sources and profiling algorithms, configure and start profiling processes, and list and visualize profiling results. It also provides access to previous profiling runs and their results so that a user can review all metadata grouped by their dataset. The frontend builds upon the Google Web Toolkit (GWT) and communicates over a RESTful API with the backend component.

Metanome is shipped with its own jetty web-server so that it runs out of the box, requiring only a JRE 1.7 or higher to be installed. No further software is required by Metanome itself, but the tool can read data from an existing database or run algorithms that utilize external frameworks, such as MATLAB.

2.2 Profiling Framework

Metanome acts as a framework for different kinds of profiling algorithms. Because most of the algorithms perform the same common tasks, Metanome provides standardized functionality for them. In the following, we discuss the four most important tasks and the provided functionality:

Input Parsing. The first task of the Metanome framework is to build an abstraction around input sources, because specific data formats, such as separator characters in CSV-files, are irrelevant for the profiling algorithms. Hence, algorithms can choose between four standardized types of inputs:

Relational. The algorithm accepts any kind of relational input. The input source can be a file or a table in a database. The input is read sequentially while Metanome performs the parsing of records depending on the actual source.

File. The algorithm accepts raw files as input. It can, then, decide to either read and parse the content itself or, if the content is relational, to use Metanome functionality for the parsing. In this way, a Metanome algorithm can read non-relational formats, such as JSON, RDF, or XML.

Table. The algorithm accepts database tables as input. The advantage of only accepting database tables is that the algorithm is able to use database functionality when reading the tables. For instance, the tables can be read sorted or filtered by some criterion.

Database. The algorithm accepts an entire database as input. It must, then, select the tables itself, but it is also able to access metadata tables containing schema and data type information that can be used in the profiling process. To do so, Metanome must provide the type of database, e.g., DB2, MySQL, or Oracle, to the algorithm as well, because the name and location of metadata tables is vendor-specific.

Output Processing. Metanome's second task is to standardize the output formats depending on the type of metadata that the algorithm discovers. This is important, because Metanome can process and automatically analyze the results if it knows their type. To build a graphical visualization of an inclusion dependency graph, for instance, Metanome must know that the output contains inclusion dependencies and it must distinguish their dependent and referenced attributes. The most important types of metadata supported currently are *unique column combinations* (UCCs), *inclusion dependencies* (INDs), *functional dependencies* (FDs), *order dependencies* (ODs), and *basic statistics*. The metadata type "basic statistics" is designed for simple types of metadata, such as minimum, maximum, average, or median that do not require individual output formats; it also captures those types of metadata that have not been implemented in Metanome, yet.

Parametrization Handling. Besides input and output standardization, Metanome also defines the parametrization of algorithms. For this purpose, the profiling algorithms need to expose their configuration variables. The variables can then be set by the user. In this way, an algorithm could ask for a maximum number of results or a search strategy option.

Temporary Data Management. Sometimes, algorithms must write intermediate results or operational data to disk, for instance if memory capacity is low. For these cases, Metanome provides dedicated temp-files. An algorithm can store its temporary data in such files, while Metanome places them on disk and cleans them when the algorithm has finished.

2.3 Profiling Algorithms

To run within Metanome, a profiling algorithm needs to implement a set of light-weight interfaces: The first interface defines the algorithm's output type and the second interface its input type as described in Section 2.2. Choosing one output and one input type is mandatory. A holistic profiling algorithm, i.e., an algorithm that discovers multiple types of metadata, might choose more than one output type. Further interfaces can optionally be added to request certain types of parameters or temp files. For instance, an algorithm could request a regular expression for input filtering using the String-parameter interface. The algorithm can also define the number of parameters. So an inclusion dependency algorithm would require multiple relations in order to discover foreign key candidates between them.

Apart from the interface, the profiling algorithms work fully autonomously, i.e., they are treated as foreign code modules that manage themselves, providing maximum flexibility for their design. So an algorithm is able to, for instance, use distributed systems like MapReduce, machine learning frameworks like Weka, or subroutines in other programming languages without needing to change the Metanome framework. This freedom is also a risk for Metanome, because foreign code can produce memory leaks if it crashes or is terminated. Therefore, algorithms are executed in separate processes with their own address spaces. Apart from memory protection, this also allows Metanome to limit the memory consumption of profiling runs. Of course, Metanome cannot protect itself against intentionally harmful algorithms, but the profiling platform is designed for a trustworthy research community. Metanome already provides algorithms for the discovery of:

- **UCCs:** DUCC [4]
- **INDs:** MIND [7], SPIDER [2], BINDER [10]
- **FDs:** TANE [5], FUN [8], FD_MINE [12], DFD [1], DEP-MINER [6], FASTFDs [11], FDEP [3]
  (see [9] for an experimental evaluation of these algorithms)
- **ODs:** ORDER [in progress]

3. PROFILING WITH METANOME

We examine Metanome from two user perspectives: an IT professional, who uses Metanome as a profiling tool on his data, and a scientist, who develops a new profiling algorithm using Metanome as a framework.

3.1 Metadata Discovery

Given an individual dependency candidate, it is easy to *validate* it on some dataset. The challenge of data profiling is to *discover* all dependencies, i.e., to answer requests such as "Show me all dependencies of type X that hold in a given dataset." Metanome is designed for exactly such requests. Therefore, an IT professional needs to specify at least a dataset and the type of dependency that should be discovered to start a profiling run.

Usually, IT professionals bring their own data – for the demo we will provide various exemplary datasets in file or database format. To profile the data, a user must register a new dataset with the data source's format, e.g., the separator, quote and escape characters on file sources or the URL, username and password for database sources. Afterwards, the new dataset appears in Metanome's list of profilable datasets. A set of profiling algorithms is already provided in Metanome by default. With the algorithm, the IT professional chooses the type(s) of metadata that are to be discovered.

Starting a profiling run is easy: Select an algorithm, choose a data source and, if needed, set some algorithm-specific parameters. During execution, Metanome measures the algorithm's runtime and reports on its progress if its implementation supports progress measurement. In case an execution lasts too long, users can safely cancel it. On termination, the results are listed in the frontend. As profiling results can be numerous, Metanome uses pagination and loads only a subset into the frontend. The list of results can then be browsed, ranked, and visualized to find interesting dependencies.

Figure 3 shows an exemplary visualization of inclusion dependencies. The bubble chart on the left indicates inclusion dependency clusters, i.e., sets of tables that are connected via INDs. These connections indicate possible foreign-key relationships and, hence, possible join paths. Clicking one cluster in the left chart opens its graph representation on the right side. Each node represents a table, and each edge one (or more) inclusion dependencies. This visualization not only allows the user to find topic-wise related tables but also shows how to link their information.

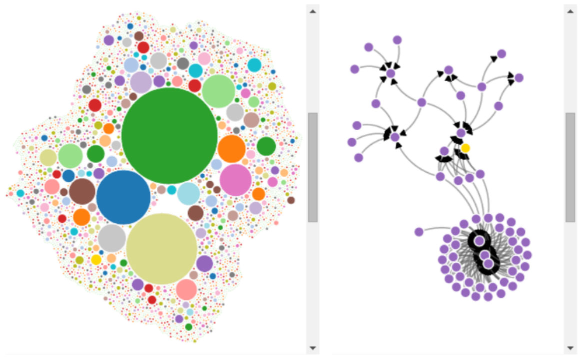


Figure 3: Visualization of inclusion dependency clusters (left) and the table join graph of one cluster (right).

3.2 Algorithm Development

To offer state-of-the-art profiling algorithms in Metanome, it must be easy for developers to integrate their work. As discussed in Section 2.3, an algorithm must specify its input type, the type of metadata that it calculates, and the parameter it needs via interfaces. The easiest way to develop a new algorithm is to use the template algorithm provided on Metanome's GitHub page and extend it[3]. In order to get the Metanome interfaces that connect the algorithm with the frameworks standardized I/O, UI, and runtime features, we recommend using Apache Maven; all Metanome components are available as Maven dependencies.

During development, Metanome supports developers by providing standard components for common tasks. But Metanome also supports the testing and evaluation of new algorithms, because a developer can easily compare the results and runtimes of her solution to previous algorithms using the same execution environment. It is also much easier for her to validate the algorithms result using Metanome's result management features.

4. SYSTEM DEMONSTRATION

In the Metanome demonstration, attendees can profile different datasets from the UCI machine learning repository[4], the Web Data Commons project[5], and others using at least 14 state-of-the-art profiling algorithms. The demonstration focuses on the most popular profiling tasks, namely inclusion dependencies, functional dependencies, unique column combinations, and order dependencies. The attendees can compare the different runtimes and discover the limits of current profiling techniques. This shows IT professionals for what kind of data they can use automated profiling methods and researchers where novel solutions are still needed.

The demo will show that already small datasets can harbor large amounts of metadata. To interpret these metadata, we introduce various result management techniques: When a profiling run has finished, the user is first provided with a list of profiling results. This list can be scrolled, filtered, and sorted by different criteria, such as lexicographical order, length, or coverage. Besides the list-based result analysis feature, attendees of our demonstration can experiment with different visualization techniques: Via collapsible prefix-trees or zoom-able sunburst diagrams (see Figure 4), for instance, users can interactively explore different result sets of unique column combinations in order to identify keys.

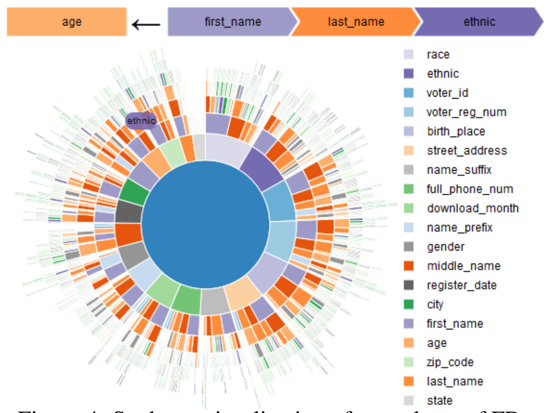


Figure 4: Sunburst visualization of a result set of FDs.

In the demo, we also show how new algorithms and new datasets can be added into the Metanome platform. Attendees can interact with Metanome using its Web frontend in order to lead their own profiling processes. While experimenting with our result management techniques, users learn for what practical tasks, e.g., primary key discovery, metadata can be used. Our goal is to show how easy the profiling of complex metadata can be and what profiling techniques are currently available. We also aim to get researches involved in this topic showing where future work is still needed, i.e., research on more efficient profiling algorithms and, due to the large metadata result sets, research on result management techniques.

[3]Since Metanome's inception, over 100 new and known algorithms have been developed as course-work, through master's theses, and by researchers. Only the best are included in the distribution.

[4]`http://archive.ics.uci.edu/ml`

[5]`http://webdatacommons.org/`

Acknowledgements

We thank Carl Ambroselli, Claudia Exeler, Tommy Neubert, Daniel Roeder, Marie Schäffer, Alexander Spivak, and Fabian Tschirschnitz for their various contributions to Metanome.

5. REFERENCES

[1] Z. Abedjan, P. Schulze, and F. Naumann. DFD: Efficient functional dependency discovery. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 949–958, 2014.

[2] J. Bauckmann, U. Leser, and F. Naumann. Efficiently computing inclusion dependencies for schema discovery. In *ICDE Workshops*, page 2, 2006.

[3] P. A. Flach and I. Savnik. Database dependency discovery: a machine learning approach. *AI Communications*, 12(3):139–160, 1999.

[4] A. Heise, J.-A. Quiané-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. *Proceedings of the VLDB Endowment*, 7(4):301–312, 2013.

[5] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[6] S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 350–364, 2000.

[7] F. D. Marchi, S. Lopes, and J.-M. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems (JIIS)*, 32(1):53–73, 2009.

[8] N. Novelli and R. Cicchetti. FUN: An efficient algorithm for mining functional and embedded dependencies. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 189–203, 2001.

[9] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10), 2015.

[10] T. Papenbrock, S. Kruse, J.-A. Quiané-Ruiz, and F. Naumann. Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment*, 8(7):774–785, 2015.

[11] C. Wyss, C. Giannella, and E. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *Proceedings of the International Conference of Data Warehousing and Knowledge Discovery (DaWaK)*, pages 101–110, 2001.

[12] H. Yao and H. J. Hamilton. Mining functional dependencies from data. *Data Mining and Knowledge Discovery*, 16(2):197–219, 2008.