DataViadotto

A GUIDE TO ENTITY/RELATIONSHIP PROFILING

Whitepaper

SUMMARY

This guide provides a brief introduction to DataViadotto Profiler. It aims at giving an overview of the available functionality and typical workflows.

DataViadotto Profiler allows users to automatically discover key and foreign key constraints by analysing relationships between data values. This is in contrast to more basic data visualization tools, which simply visualize constraints explicitly defined. It can thus be employed in cases where constraints are not fully known, or not known at all. Some common use cases are:

- schema discovery
- schema optimization
- data lake traversal
- data integration

Inputs are currently limited to relational data, but a variety of storage formats such as relational databases or CSV files are supported.

1. BACKGROUND

A key point of distinction between our and other data profiling tools lies in how it deals with incomplete or dirty data.

Incomplete data

When profiling data sets with missing values, proper handling of missing (null) values is critical. For example, treating null values as any other domain value can cause valid constraints to be missed. DataViadotto Profiler supports multiple semantics in dealing with them, which will be described later on.

Dirty data

When constraints are not actively enforced, they tend to get violated over time. Thus it becomes important to search for constraints that *almost* hold. Our profiling tool uses multiple metrics to measure the degree of constraint satisfaction, which can be used for filtering. The optimal parameters to use here vary between data sets, and typically require some experimentation to find.

We note that data can be dirty in a variety of ways (e.g. inconsistent, outdated, or simply wrong). For constraint discovery, only dirtiness that causes data inconsistency matters.

Sampling

The first step in profiling a data set is sampling, where a suitable subset of records is selected from each table. The main purpose of this step is to speed up constraint discovery. Additionally, basic characteristics such as numbers of null and distinct values for each column are computed as part of the sampling process, which are needed during profiling. For this reason, constraint discovery will *always* operate on samples.

For tables with few records, all records will be included in the sample. For larger tables the *number* of records sampled increases with the table size, but the *fraction* of records sampled becomes smaller as the table grows. Although sampling will inevitably result in some inaccuracies for the measures computed, sampling rates are chosen to ensure that these inaccuracies remain small.

It is possible to force all records to be included in a sample – doing so will avoid inaccuracies, but at the cost of reduced speed and increased memory usage.

2. MINING KEYS

Our tool currently supports two profiling operations for keys:

- **Key Discovery**: Find all minimal column sets that form a key for a given table.
- **Key Analysis**: Compute uniqueness metrics for a given column set, and extract data examples.

For key discovery, we note that just because a constraint happens to hold for a table does not guarantee that it is meaningful. In practice, many of the key constraints discovered will be accidental, meaning they hold only by chance – this is particularly true for tables with few records.

As accidental keys are typically of little interest, the set of minimal keys returned is best seen as a starting point for further manual evaluation by a domain expert. Key analysis for selected column sets can be helpful in this.

Incomplete Data

There are different approaches for dealing with null values. Conceptually we consider *possible worlds*, where missing values are replaced with arbitrary values. Depending on whether we require a key constraint to be satisfied by all possible worlds or only some, we obtain different semantics.

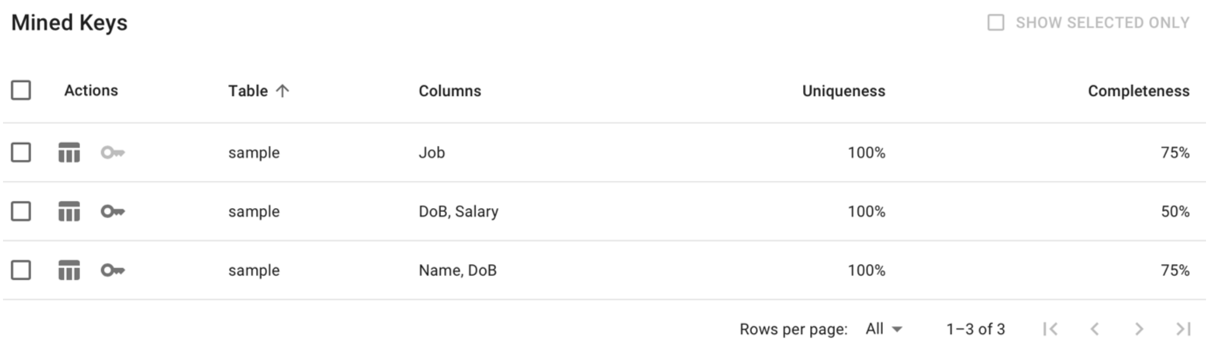
The default is *possible semantics*, corresponding to UNIQUE constraints in SQL. Here a column set **K** is a key if no two records violate it, meaning they have matching values on all columns in **K**, and none of these values is missing. Thus the column set {Name, DoB} is a key for the table below, but {Name, Salary} is not a key.

Name	DoB	Salary	Job
John	01/03/85	90k	Developer
John		90k	Analyst
Susan	01/03/85		
Dave	06/06/91	75k	Tester

Note that {DoB, Salary} and {Job} are also keys for the given table, though likely accidental.

Since only keys are returned that hold on the given table, their *uniqueness ratio*, the percentage of records that can be uniquely identified by the columns of the key or contain a null value on some column of the key, will always be 100%.

We may assess the effectiveness of keys under possible semantics by their *completeness ratio*, the percentage of records that have no null value in any column of the key. Hence, the completeness ratio says what percentage of records can be identified uniquely by the key.



Mined Keys

☐ SHOW SELECTED ONLY

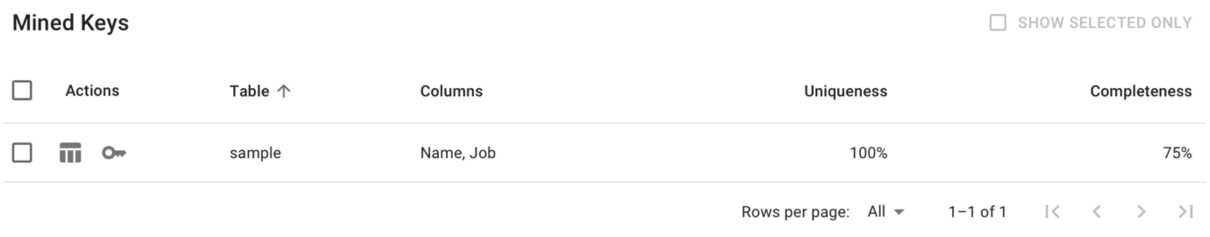
☐	Actions	Table ↑	Columns	Uniqueness	Completeness
☐		sample	Job	100%	75%
☐		sample	DoB, Salary	100%	50%
☐		sample	Name, DoB	100%	75%

Rows per page: All 1–3 of 3

Fig. 1: Keys under possible semantics mined from sample

Fig. 1 shows the results of mining keys from the sample table above.

Under *certain semantics* each pair of records must strongly disagree on at least one key column, meaning values are present and different. For the table above, {Name, Job} is the only minimal key. The completeness ratio of a certain key does not indicate its effectiveness. Indeed, certain keys are always able to uniquely identify every record of the given table.



Mined Keys

☐ SHOW SELECTED ONLY

☐	Actions	Table ↑	Columns	Uniqueness	Completeness
☐		sample	Name, Job	100%	75%

Rows per page: All 1–1 of 1

Fig. 2: Keys under certain semantics mined from sample

For more information on possible and certain SQL keys, see an academic paper where we introduced the concept.

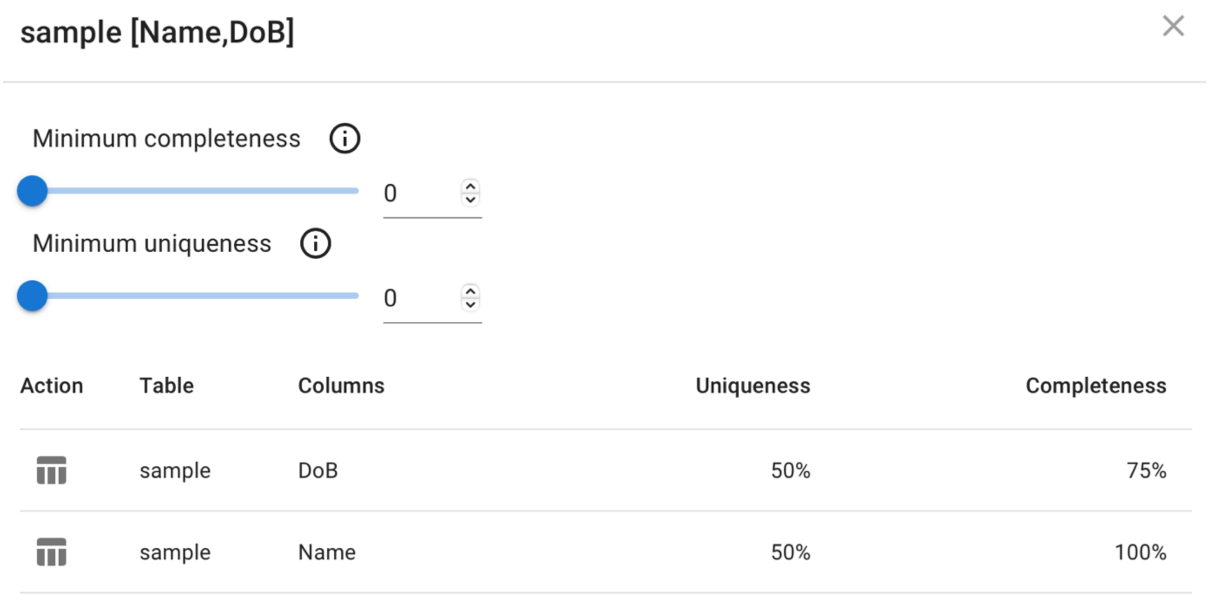
Dirty Data

Key analysis computes the *uniqueness ratio* of a given set of columns. This is the percentage of records which do not violate the key constraint (together with some other record). For example, given the table above, the uniqueness coefficient of {Name} is 50%.

Generally, uniqueness ratios close to 100% indicate that the column set might be a key, with a few dirty data values causing its violation. For low uniqueness ratios this is unlikely.

For incomplete data we compute the minimum and maximum uniqueness ratio amongst all possible worlds. For example, given the table above, the uniqueness ratio of {Name, DoB} varies between 50% and 100%. Keys under possible semantics have a maximum uniqueness ratio of 100%, while keys under certain semantics have a minimum uniqueness ratio of 100%.

Another interesting feature is the ability to show uniqueness and completeness ratios for all subsets of a mined key. Since all keys returned by our mining algorithms are minimal, removal of any column will result in a proper subset whose uniqueness ratio must be smaller than 100%. That means, some records have matching non-null values on all columns of the subset. If the subset represent a meaningful key, any records that participate in a violation of this subset need to exhibit some inconsistency. For instance, Fig. 3 shows an analysis of all subsets for the key {Name, DoB}.


sample [Name,DoB]

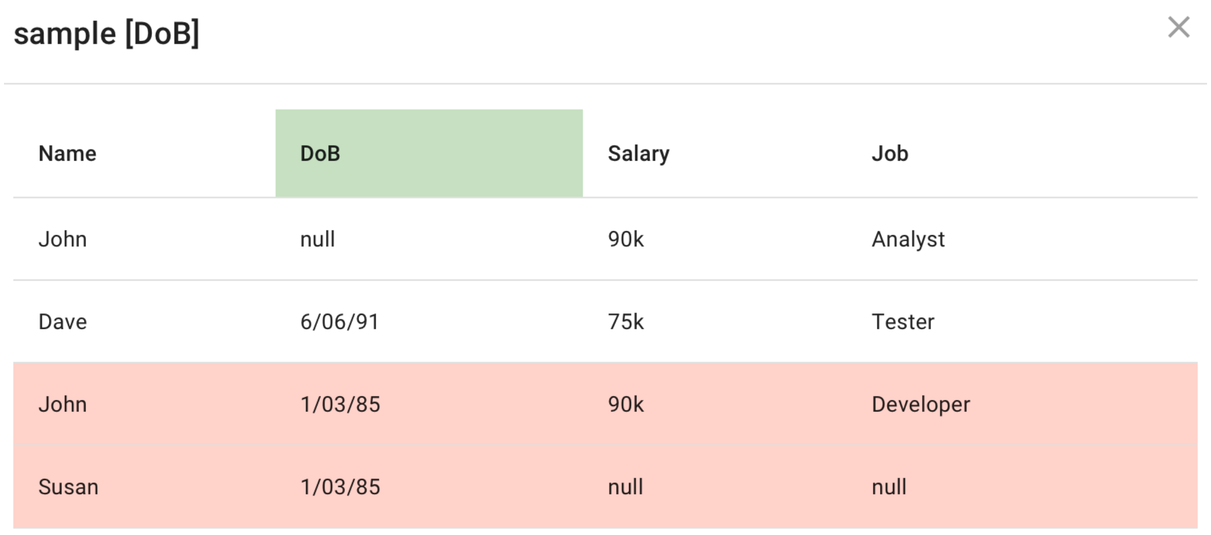
Minimum completeness 0

Minimum uniqueness 0

Action	Table	Columns	Uniqueness	Completeness
	sample	DoB	50%	75%
	sample	Name	50%	100%

Fig. 3: Analysis of all subsets for key {Name, DoB}

An important feature in analysing the sensibility of a mined key or its subset is the inspection of example data. Here, users can see example records that confirm the validity of the constraint under inspection. In case the uniqueness ratio is not 100%, pairs of sample records are also shown that invalidate the constraint. In case the completeness ratio is not 100%, sample records that contain a null value in some key column are shown. For instance, Fig. 4 shows example data for the key {DoB}.



sample [DoB]

Name	DoB	Salary	Job
John	null	90k	Analyst
Dave	6/06/91	75k	Tester
John	1/03/85	90k	Developer
Susan	1/03/85	null	null

Fig. 4: Example records illustrating uniqueness and completeness ratios of the key {DoB}

3. MINING FOREIGN KEYS

Rather than just focusing on foreign keys, our tool searches for *inclusion dependencies* (INDs), meaning that the column set referenced does not need to be a key. However, as foreign keys are the most common type of IND, we provide options for filtering out INDs where the referenced column set is not a key.

Our tool supports two profiling operations for INDs/FKs:

- **Foreign Key Discovery**: Given a set of tables, identify INDs / FKs between them.
- **Foreign Key Analysis**: Compute metrics for a given IND, and extract data examples.

As for key discovery, foreign key discovery will return many accidental INDs, so results returned will likely require further evaluation by a domain expert. Foreign Key Analysis can help with this.

Note that foreign key discovery also returns non-maximal INDs. This is redundant when dealing with exact INDs only. However, when data sets are dirty, the degree of satisfaction tends to be greater for non-maximal INDs, which may make them more relevant.

Incomplete Data

Different semantics are supported for deciding how missing values are handled for inclusion dependencies. That is, if a record in the source table is missing a value in one of the columns participating in the IND, then satisfaction of the IND for that records depends on the semantics.

- **Simple Semantics**: The IND is satisfied.

- **Partial Semantics**: The IND is satisfied if there is some record in the target table that matches the record in the source table for every column pair in the IND where the source record has a non-null value.
- **Full Semantics**: The IND is violated.

FOREIGN KEY constraints in SQL use simple semantics. Thus for schema discovery and optimization tasks where FOREIGN KEY constraints were previously enforced (but are now unavailable for some reason), simple semantics can be an appropriate choice.

Partial semantics can be a sensible choice for most tasks, as it best captures the intent of an inclusion dependency (references must target existing data) and should be considered the default.

For some datasets, references will almost always be complete (no missing values). Here full semantics is best at eliminating false positives, as it is the most restrictive.

We note that for a column containing only null values, any IND with such a column as source would technically be satisfied under simple and partial semantics. However, we exclude such columns from the mining process, as the resulting INDs would not be useful.

Consider the following two tables, Orders and Accounts.

OrderID	Customer	Company
1	Darrel	Bugs-R-Us
2	Susan	
3	Edward	
4	Jenny	
5	Tom	EazyCode

Name	Company	AccountNo
Darrel	Bugs-R-Us	99-666-00
Susan		71-922-88
Edward	Bugs-R-Us	99-666-00
Darrel	EazyCode	12-345-67
Tom		57-902-46

The inclusion dependency *Orders[Customer,Company]* ⊆ *Accounts[Name,Company]* is satisfied for the first order under all semantics. Orders 2 and 3 satisfy it under simple and partial semantics, but not under full semantics. Order 4 satisfies it under simple semantics only, while order 5 violates it under all semantics.

Dirty Data

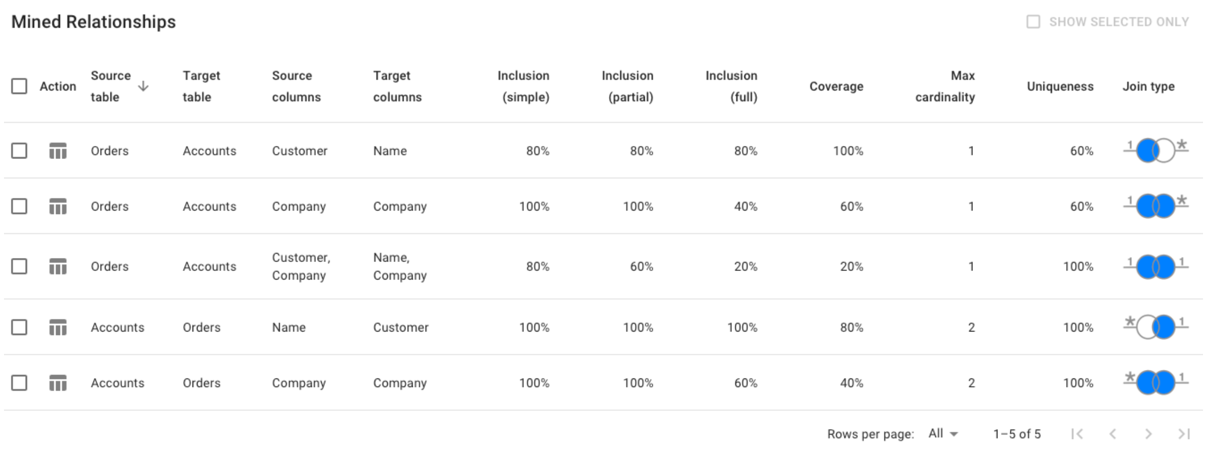
The inclusion ratio of an IND measures the degree to which a data set satisfies it. It is computed as the percentage of records in the source table that satisfy the IND. As satisfaction depends on the choice of semantics, we obtain multiple inclusion coefficients, one per semantics. For example, given the two tables above, the inclusion coefficient for *Orders[Customer,Company]* ⊆ *Accounts[Name,Company]* is 80% under simple semantics, 60% under partial semantics, and 20% under full semantics.

During foreign key discovery and analysis, inclusion ratios (IRs) are computed with respect to all three semantics. For example, given the two tables above, foreign key discovery will find

- *Orders[Customer]* ⊆ *Accounts[Name]* with IRs of 80%, 80% and 80%
- *Orders[Company]* ⊆ *Accounts[Company]* with IRs of 100%, 100% and 40%

- *Orders[Customer,Company] ⊆Accounts[Name,Company]* with IRs of 80%, 60% & 20%.

Fig. 5 shows the INDs mined between the Orders and Accounts tables, including their IRs, other measures and an illustration of the best type of join available for each IND between the two tables.



Mined Relationships

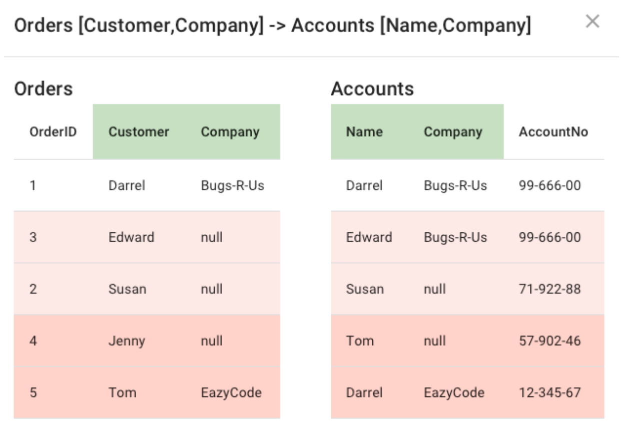
☐ SHOW SELECTED ONLY

☐	Action	Source table ↓	Target table	Source columns	Target columns	Inclusion (simple)	Inclusion (partial)	Inclusion (full)	Coverage	Max cardinality	Uniqueness	Join type
☐	▦	Orders	Accounts	Customer	Name	80%	80%	80%	100%	1	60%	1 ●○ *
☐	▦	Orders	Accounts	Company	Company	100%	100%	40%	60%	1	60%	1 ●● *
☐	▦	Orders	Accounts	Customer, Company	Name, Company	80%	60%	20%	20%	1	100%	1 ●● 1
☐	▦	Accounts	Orders	Name	Customer	100%	100%	100%	80%	2	100%	* ○● 1
☐	▦	Accounts	Orders	Company	Company	100%	100%	60%	40%	2	100%	* ●● 1

Rows per page: All ▾ 1–5 of 5 |< < > >|

Fig. 5: Mined INDs between Orders and Accounts tables with different IRs and other measures

An inspection of sample records for pairs of tables is a helpful tool for users who need to decide which INDs represent meaningful business rules, and which semantics is appropriate for them to use. Fig. 6 shows a data example for the IND *Orders[Customer,Company] ⊆ Accounts[Name,Company]*. Records on white satisfy full semantics, records on light red satisfy partial but not full semantics, and records on dark red violate partial semantics. In Fig. 6, the record with OrderID 4 satisfies simple semantics, while the record with OrderID 5 does not even satisfy simple semantics.



Orders [Customer,Company] -> Accounts [Name,Company] ✕

Orders

OrderID	Customer	Company
1	Darrel	Bugs-R-Us
3	Edward	null
2	Susan	null
4	Jenny	null
5	Tom	EazyCode

Accounts

Name	Company	AccountNo
Darrel	Bugs-R-Us	99-666-00
Edward	Bugs-R-Us	99-666-00
Susan	null	71-922-88
Tom	null	57-902-46
Darrel	EazyCode	12-345-67

Fig. 6: Sample records from Orders and Accounts to illustrate the IRs of the IND under inspection

Filtering

Without restrictions, foreign key discovery is likely to return an overwhelming number of false positives – INDs with non-zero inclusion ratios that are not meaningful – as the number of potential INDs grows exponentially in the number of columns. This can also result in timeouts or memory overflows, even for small datasets. Columns with small integer values are particularly prone to this, as they are likely to have at least some values in common.

To avoid this, INDs must be filtered. The most obvious way to do this is by defining thresholds for the different inclusions ratios, so that only INDs whose inclusion ratios meet or exceed these thresholds are returned. Thresholds for partial and full semantics are used for pruning during the discovery process, while the threshold for simple semantics is only applied afterwards. Thus higher thresholds for partial and full semantics will speed up the discovery process, while a higher threshold for simple semantics will not.

While an inclusion dependency does not require all records in the target table to be referenced, in practice they often are. For example, if every order contains at least one item, then every OrderID value will be referenced by an OrderItem. The *coverage* measure describes the percentage of records in the target table that are referenced by one or more records in the source table under full semantics. Filtering INDs by coverage is particularly helpful for eliminating false positives between ID columns which had values auto-populated from 1 to n, leading to accidental INDs with high inclusion ratios. However, as meaningful INDs may have low coverage (e.g. Managers referencing Persons), this must be used with care. It can be sensible to set only low coverage thresholds, and use coverage mainly to guide manual evaluation.

4. COMMON WORKFLOWS

While workflows for data profiling will depend on how data sets are being processed, some sequences of steps arise frequently. We list some of these below. It is assumed that data sets have already been sampled, as this is required for all further processing.

Mine keys -> curate -> mine foreign keys

Mining foreign keys directly can result in long processing times and large numbers of false positives. By specifying a set of target keys for each table (possibly empty), we can address both of these issues. We can obtain these target keys by first mining for keys, then manually curating the keys found. If keys are of interest, this may be necessary anyway. Note that this will restrict inclusion dependencies found to foreign keys, which may not always be desirable.

Mine with low thresholds -> explore -> tighten thresholds

By specifying the right thresholds for a data set during mining, one can strike a good balance between meaningful constraints found, false positives eliminated, and processing time.

However, which thresholds are the “right” ones varies between data sets, and tasks performed. It thus becomes necessary to develop some intuition of which thresholds work for filtering out unwanted constraints.

We do this by first mining with low thresholds, to ensure meaningful constraints are not missed. Once a better understanding has been obtained by examining the constraints discovered, thresholds can be tightened. Note that tightening of thresholds can be done by filtering locally, that is, the mining algorithm does not need to be invoked again.

Mine with high thresholds -> loosen thresholds -> explore -> tighten thresholds

This extension of the previous workflow becomes necessary if mining with low thresholds turns out to be too expensive, resulting in timeouts or memory overflows. Here we start with high thresholds to ensure we get some results quickly, then loosen them gradually, until a good balance between capturing meaningful constraints and processing time is reached. Note that this will require the mining process to be invoked again. Afterwards we proceed as before, exploring constraints discovered and tightening thresholds to eliminate false positives.

5. CLOSING

In summary, DataViadotto Profiler offers unique features that empower data professionals to distinguish meaningful keys and inclusion dependencies from accidental ones. These include methods that analyze the validity of such constraints under dirty data and different interpretations of incomplete data. The understanding of users is further enhanced by carefully chosen example records of data that illustrate the degree of validity for any constraint under inspection. Exploring different thresholds for our measures allows users to pivot towards the right balance between precision and recall for those constraints that govern the data sets.

ABOUT DATAVIADOTTO

DataViadotto is the industry pioneer for Entity/Relationship profiling technology. The company draws on decades of academic research in the subject to make the process of discovering models from data more effective, efficient and intuitive. Ultimately, data becomes profitable.

FOR ADDITIONAL QUESTIONS, CONTACT DATAVIADOTTO

www.viadotto.tech