DPQL: Applications for Holistic Data Profiling

Marcian Seeger [1] and Thorsten Papenbrock [2]

Abstract: Data profiling is the process of extracting implicit metadata, such as data types, attribute statistics, and various types of data dependencies, from raw datasets. Because structural metadata is often not stored explicitly, many data management applications rely on data profiling to identify, for example, functional dependencies, inclusion dependencies, or unique column combinations. The utilization of automatically discovered metadata within use cases, such as data discovery, integration, normalization, cleaning, or optimization, is however a still very complicated process. This is because existing data profiling approaches consider different types of metadata in isolation although in practice many use cases require specific combinations, i.e., patterns of structural metadata. For this reason, we recently proposed the Data Profiling Query Language DPQL that can express (and in the near future holistically discover) arbitrary patterns of various types of metadata. DPQL query patterns allow data scientists to express exactly what metadata real-world applications require. This eliminates otherwise complicated post-processing efforts, reduces result sizes, and might lead to significantly shorter profiling times. To demonstrate the expressiveness and versatility of DPQL, we survey a variety of data engineering applications in this paper and solve their metadata requirements with concrete DPQL query patterns. We also measure DPQL result sizes on two benchmark datasets and compare them to the result sizes of standard data profiling algorithms to demonstrate that holistic data profiling with DPQL produces more suitable and in many cases also much smaller result sets.

Keywords: data profiling, holistic discovery, functional dependencies, unique column combinations, inclusion dependencies, order dependencies

1 Data Profiling and Application Requirements

Structural metadata are rules and constraints that describe the shape and format of datasets. They make datasets accessible, provide guidelines for the evolution of the datasets, and model correctness-defining dataset properties. For this reason, structural metadata, such as data types, value domain models, and data dependencies, are essential to many data management processes ranging from data exploration [Fe18; Ro09] over data integration [DR02; Zh10] and data cleaning [IC15; VA18] to query optimization [KPN22; Pa00] and machine learning [Ch19; KPN20] to name only a few. In practice, though, most types of structural metadata are rarely stored explicitly with the data that they refer to. This is particularly true for data dependencies, such as functional dependencies (FDs) [Co71], inclusion dependencies (INDs) [CFP82], order dependencies (ODs) [SGG12], and unique column combinations (UCCs) [LO78]. For this reason, data scientists and data engineers perform *data profiling* [Ab18] to discover structural metadata from already existing dataset instances.

[1] University of Marburg, Big Data Analytics, Hans-Meerwein-Str. 6, 35032 Marburg, Germany, marcian.seeger@uni-marburg.de, https://orcid.org/0009-0001-8483-2212

[2] University of Marburg, Big Data Analytics, Hans-Meerwein-Str. 6, 35032 Marburg, Germany, papenbrock@informatik.uni-marburg.de, https://orcid.org/0000-0002-4019-8221

Modern data profiling algorithms support these efforts as they are able to automatically discover complete sets of usually minimal or maximal data dependencies of a specific type. For the previously mentioned data dependencies, for example, various algorithms exist that can profile FDs [Bl24; Pa15a], INDs [Dü19; Pa15b], ODs [Ch23; Li24; SP22], and UCCs [Bi20; PN17a] efficiently. Unfortunately, though, many applications require combinations of different metadata statements and, hence, cannot work with the profiling results directly. To (re-)discover foreign key relationships in a relational database, for example, we need to search for inclusion dependencies `X` $\subseteq$ `Y` with a unique column combination `Y` on the referenced side for referential integrity. Forming such combinations with the minimal/maximal profiling results of existing data profiling solutions is not trivial, because the data dependencies in the patterns (here the IND) are not necessarily minimal/maximal. For this reason, we recently proposed the *Data Profiling Query Language* (DPQL) [Se23] that allows the declarative specification of arbitrary patterns of primitive data dependencies.

A DPQL query is a declarative **SELECT**-**FROM**-**WHERE** statement similar to SQL, but it targets result tuples formed of attribute lists, which we call *column combinations*, and not instance records. List. 1 shows an example of a DPQL query that asks for foreign key relationships between the tables `Actors`, `Movies`, `Producers`, and `Studios`. The **FROM**-clause specifies that the foreign key `X` and the key `Y` can be arbitrary column combinations (**CC**) from either of the four tables. Technically, the DPQL **CC**-operator describes all possible attribute combinations that can be created with the attributes of the schemata provided as its arguments. The **WHERE**-clause enforces the IND `X` $\subseteq$ `Y` with the DPQL operator **IND**`(X,Y)` and the UCC `Y` with the operator **UCC**`(Y)`. DPQL also offers operators for other primitive dependencies, such as FDs `X` $\rightarrow$ `Y` (**FD**`(X,Y)`), ODs `X` $\rightarrow_{\leq}$ `Y` (**OD**`(X,Y)`), or MvDs `X` $\twoheadrightarrow$ `Y` (**MvD**`(X,Y)`). Besides the primitive dependencies, the **WHERE** clause can enforce further properties, such as **SPLIT**`(X,Y)`, which restricts `X` and `Y` to different relations, **SIZE**`(Y) <= 2`, which requires `Y` to contain at most two attributes, **CONTAINS**`(X,Y)`, which makes `X` a superset of the attributes of `Y`, or **COALESCE**`(X,Y)`, which restricts the attributes of `X` and `Y` to the same relation. All these filters can be used to refine the base patterns of primitive dependencies. In the remainder of this paper, we investigate specific data dependency patterns, such as the IND-UCC-combination in List. 1, and abstract from the projections in the **SELECT** clauses, the concrete instances in the **FROM** clauses, and optional filters in the **WHERE** clauses. A DPQL query by default always returns minimal dependency combinations, which means that *no attributes can be removed from a result statement without invalidating the statement*. This minimality holds for an entire dependency combination, such that the contained primitive dependencies may be non-minimal.

```
1  SELECT
2    X AS ForeignKey, Y AS Key
3  FROM
4    CC(Actors,Movies,Producers,Studios) X,
5    CC(Actors,Movies,Producers,Studios) Y
6  WHERE
7    IND(X,Y) AND UCC(Y)
8    AND SPLIT(X,Y) AND SIZE(Y) <= 2
```

List. 1: Find all foreign key candidates between the tables `Actors`, `Movies`, `Producers`, and `Studios`.

DPQL is a flexible metadata query language that supports arbitrary patterns of primitive data dependencies. So far, however, data scientists have made do with existing profiling techniques so that the advantages of DPQL for real-world applications may not be apparent yet. For this reason, we present 31 example queries for different data science applications. We provide short descriptions of the various applications in Sect. 2 and evaluate the queries by comparing DPQL result sizes to the result sizes of state-of-the-art profiling runs in Sect. 3. This paper is an invitation to researchers in different data science disciplines to participate in the DPQL research; it should also accelerate the adoption of DPQL in the discussed applications once our first fully functional DPQL engine is released.

2 Applications

Data profiling has numerous applications in data engineering, including data linkage, normalization, cleaning, and integration scenarios. Statistics and other forms of structural metadata about relational datasets are also useful for data management applications, which include schema and integrity maintenance, data provenance, and query optimization. Further applications can be found in the area of data analytics and include, inter alia, feature engineering and explainability research. Many use cases for structural metadata have already been summarized and surveyed in related work [SC23; So20], but the combination of individual data dependencies has received little attention so far, presumably because they have been harder to express.

In this chapter, we focus on a subset of applications with very specific needs for holistic data profiling results and present concrete DPQL query patterns that meet these requirements. To demonstrate the versatility of DPQL, we collected 31 representative use cases and formulated suitable DPQL queries that solve their metadata needs (see Tab. 1). A few duplicate query patterns in the listing serve multiple different use cases. We further grouped the applications into seven application domains. In the following sections, we describe the applications and their queries for each domain in more detail. Please note that profiled dependency combinations represent *candidates* for foreign keys, redundant features, common effects etc. but still usually require further domain-specific evaluation.

Data Linkage (DL) Data linkage is an application area related to data integration with the purpose of connecting different data sources through the discovery of possible join paths. In relational databases, join paths are formed by (implicit or explicit) key and **Foreign Key** (DL-1) relationships. Foreign keys are schema constraints that ensure referential integrity by establishing a link between two relational tables [Co70]. Various research projects have shown how data profiling can be used to discover both primary and foreign keys [JN20; PN17b; Ro09]. In particular, Jiang et al. proposed the holistic algorithm HoPF that uses INDs and UCCs jointly. To discover foreign keys with DPQL, we use the same general ideas when formulating the query pattern as `IND(X,Y) AND UCC(Y)`. This pattern discovers an

Ref.	Application	Query
DL-1	Foreign Key	`IND(X,Y) AND UCC(Y)`
DL-2	Embedded Link	`FD(X,Z) AND IND(X,Y) AND UCC(Y)`
DL-3	Embedded-Embedded Link	`FD(X,Z) AND IND(X,Y) AND FD(Y,W)`
SN-1	2NF Violation	`FD(X,Y) AND UCC(Z) AND CONTAINS(Z,X) AND Z <> X`
SN-2	BCNF Violation	`FD(X,Y) AND NOT UCC(X)`
SN-3	4NF Violation	`MvD(X,Y) AND NOT UCC(X)`
SN-4	IDNF Violation 1	`IND(X,Y) AND IND(Y,X) AND SPLIT(X,Y)`
SN-5	IDNF Violation 2	`IND(X,Y) AND FD(X,Y)`
SN-6	Redundant Normalization	`UCC(X) AND IND(X,Y) AND UCC(Y)`
QO-1	Foreign Key Rule	`IND(X,Y) AND UCC(Y)`
QO-2	Transitive Join Rule	`IND(X,Y) AND IND(Y,Z) AND UCC(Y)`
QO-3	Theta Join Rule	`OD(X,Y) AND UCC(X)`
QO-4	Join Simplification Rule	`IND(X,Y) AND FD(Z,W) AND CONTAINS(Y,Z)`
QO-5	Cardinality Propagation Rule	`FD(X,Y) AND FD(Y,X)`
QO-6	Uniqueness Tracking Rule	`UCC(X) AND IND(X,Y) AND UCC(Y)`
QO-7	Join-to-Predicate Rewrite Rule	`IND(X,Y) AND OD(Y,Z) AND UCC(Y)`
ML-1	Redundant ML Feature	`FD(X,Y) AND FD(Z,X)`
ML-2	Cross-Table Redundant ML Feature	`FD(X,Y) AND IND(X,W) AND FD(Z,W)`
DA-1	Common Cause	`FD(X,Y) AND FD(X,Z)`
DA-2	Common Effect	`FD(X,Y) AND FD(Z,Y)`
DA-3	Cross-Table Common Cause	`FD(X,A) AND IND(X,Y) AND FD(Y,B)`
DA-4	Cross-Table Common Effect	`FD(A,X) AND IND(X,Y) AND FD(B,Y)`
SM-1	Uniqueness Match	`IND(X,Y) AND UCC(X) AND UCC(Y) AND SPLIT(X,Y)`
SM-2	Dependent Match	`IND(X,Y) AND FD(X,Z) AND FD(Y,W) AND SPLIT(X,Y)`
SM-3	Determinant Match	`IND(X,Y) AND FD(Z,X) AND FD(W,Y) AND SPLIT(X,Y)`
SM-4	Included Match	`IND(X,Y) AND IND(Z,X) AND IND(W,Y) AND SPLIT(X,Y)`
SM-5	Including Match	`IND(X,Y) AND IND(X,Z) AND IND(Y,W) AND SPLIT(X,Y)`
SA-1	3-Clique	`IND(X,Y) AND IND(Y,Z) AND IND(Z,X)` `AND SPLIT(X,Y) AND SPLIT(Y,Z) AND SPLIT(Z,X)`
SA-2	n-Clique	`IND(X,Y) AND ... IND(Z,X)` `AND SPLIT(X,Y) AND ... SPLIT(Z,X)`
SA-3	Hub	`IND(X,Y) AND IND(Z,W)` `AND COALESCE(X,Z) AND SPLIT(X,Y) AND SPLIT(Z,W)`
SA-4	Authority	`IND(X,Y) AND IND(Z,Y)`

Tab. 1: A listing of 31 applications from seven domains and their metadata requirements expressed as DPQL query patterns.

IND between the attributes `X` and `Y`, where `Y` is a UCC. The IND ensures that every value in `X` exists in `Y` e. g. a foreign key condition, while the UCC on `Y` guarantees that `Y` is a candidate key. Additional conditions in the pattern, such as **`SPLIT`** or **`CARDINALITY`**, can make the query more precise. A standard foreign key `X` would be used to pull information from a relation with key `Y` to entities modeled in `X`'s relation. We can reverse the principle if `X` is the left-hand side of an FD $X \rightarrow Z$ and bring the right-hand side attributes `Z` to the relation of `Y`. The FD $X \rightarrow Z$ can be seen as an embedded entity in a not fully normalized database, whose attributes can also belong to the relation of `Y`. To detect such an **Embedded Link** (DL-2), we use the DPQL query pattern **`FD(X,Z) AND IND(X,Y) AND UCC(Y)`**. In an even more complex setting, an **Embedded-Embedded Link** (DL-3) might connect two embedded entities $X \rightarrow Z$ and $Y \rightarrow W$. We can find their link with the query **`FD(X,Z) AND IND(X,Y) AND FD(Y,W)`**.

Schema Normalization (SN) Schema normalization is an application domain that describes the structuring of relational database schemata in accordance with certain normal form specifications. Normal forms are means to reduce data redundancy and improve data quality, integrity, and clarity [Co70]. They provide structured approaches for normalization based on specific decomposition rules. These rules often involve finding certain combinations of FDs and other data dependencies that violate a normal form. For this reason, they can effectively be expressed as DPQL queries. For example, the second normal form (2NF) specifies that a relation should not have an FD $X \rightarrow Y$ with `X` being a real subset of a primary key `Z`, i.e., $X \subset Y$. To find such a **2NF Violation** (SN-1), we can use the DPQL query pattern **`FD(X,Y) AND UCC(Z) AND CONTAINS(Z,X) AND Z <> X`**. The Boyce-Codd Normal Form (BCNF) [BB79; Co74; PN17b] is one of the most popular normal forms demanding that every non-trivial FD $X \rightarrow Y$ has a key `X` as its determinant. To detect a **BCNF Violation** (SN-2), we can, therefore, use the query **`FD(X,Y) AND NOT UCC(X)`** that finds FDs whose left-hand side `X` does not satisfy the necessary key property. An even stricter normal form is the Fourth Normal Form (4NF) [Fa77], which is based on multi-valued dependencies (MvD). A **4NF Violation** (SN-3) is the occurrence of an MvD that is not dependent on the key, which can be detected with the DPQL pattern **`MVD(X,Y) AND NOT UCC(X)`**. The Inclusion Dependency Normal Form (IDNF) [LV00] (SN-4 and SN-5) requires a database schema to be in BCNF and that no transitivity through INDs is present. We can discover such forbidden transitivities with the DPQL patterns **`IND(X,Y) AND IND(Y,X) AND SPLIT(X,Y)`** and **`IND(X,Y) AND FD(X,Y)`**. Despite the usefulness of normalized schemata, sometimes normalization also goes too far – either because a schema was modeled overly detailed or automatic normalization is performed too aggressively. In these cases, we observe that an IND links two primary keys, i.e., same entities in different relations. Such a **Redundant Normalization** (SN-6) can be found with the DPQL query **`UCC(X) AND IND(X,Y) AND UCC(Y)`**; the join of `X` and `Y` will not produce redundant values.

Query Optimization (QO) For fast query answering, database management systems rely on efficient query optimization and execution techniques. Query optimization specifically

serves the purpose of improving query execution times through rule-based query rewriting techniques, cost and cardinality estimation, and cost-based query transformation [KPN22]. While some optimizations can be achieved using simple statistics [Ch98], more complex optimizations require the existence of certain data dependencies [KPN22]. Kossmann et al. showed that query optimization techniques based on (combinations of) data dependencies can reduce query processing times by up to 27% [Ko22]. Furthermore, Linder et al. demonstrated that some data dependencies-based query optimization techniques can lead to throughput improvements between 5% and 33% [LRN24]. In fact, the survey on data dependency-based query optimization techniques by Kossmann et al. showed that many techniques actually require combinations of multiple dependencies: For example, a **Foreign Key Rule** (QO-1), which is essentially a simple foreign key relationship identifiable as `IND(X,Y) AND UCC(Y)`, can be used for different optimizations and transformations including outer relation join optimization, semi-join simplification, invisible joins, subquery to join query, and cardinality estimation [KPN22]. Furthermore, certain joins can be eliminated with the **Transitive Join Rule** (QO-2) that requires `IND(X,Y) AND IND(Z,Y) AND UCC(Y)`; due to the transitivity of the two INDs, `Z` can be joined with `X` directly. Another query re-writing optimization uses the **Theta Join Rule** (QO-3) that requires an order dependency (OD) in combination with a UCC. This pattern can be queried as `OD(X,Y) AND UCC(X)`. The **Join Simplification Rule** (QO-4) relies on a combination of an IND and an FD and can be queried as `IND(X,Y) AND FD(Z,W) AND CONTAINS(Y,Z)`. It allows us to re-write the join predicate `Y` as $Y \setminus W$ [KY83]. The **Cardinality Propagation Rule** (QO-5) uses the cyclic FDs pattern `FD(X,Y) AND FD(Y,X)` to propagate cardinality information from `X` to `Y` and vice versa [KPN22]. Similarly, the **Uniqueness Tracking Rule** (QO-6), which is queried as `UCC(X) AND IND(X,Y) AND UCC(Y)`, helps the system to assert that a join on $X = Y$ will produce a result in which both `X` and `Y` remain UCCs; furthermore, the result has the same cardinality as `X` [PL10]. Lastly, the **Join-to-Predicate Rewrite Rule** (QO-7) can improve query execution times by up to 15% (QO-7) [LRN24]. For this, a certain pattern with an IND, OD and UCC is needed, which can be queried as `IND(X,Y) AND OD(Y,Z) AND UCC(Y)`.

Machine Learning (ML) A machine learning model is a statistical algorithm based on a mathematical model that transforms input features into (one or more) output features. To avoid, inter alia, performance, accuracy [HL23; Hu04] and overfitting issues [Sa21] caused by unnecessarily high numbers of input features, feature selection techniques are being used to identify non-indicative and redundant features [HTD09]. The **Redundant ML Feature** query (ML-1) `FD(X,Y) AND FD(Z,X)` can assist these efforts by identifying candidate attributes `X` for redundancy when predicting `Y`. Due to `FD(X,Y)`, `X` may be a good feature for `Y`, but because of `FD(Z,X)`, the features `Z` may simply subsume `X`. We might, for example, find the chain of FDs $\{DateOfBirth\} \rightarrow \{Age\} \rightarrow \{Privileges\}$, which marks `Age` as possibly redundant, or the FD chain $\{GeoX,GeoY\} \rightarrow \{Country, Postcode\} \rightarrow \{Temperature\}$, which marks *Country* and *Postcode* as possibly redundant. These `X` column combinations still need to be tested for statistical correlation or mutual information with existing feature reduction techniques, but they help the algorithms to identify *candidates* for the filtering

process and provide provenance information to better explain feature reductions. The **Cross-Table Redundant ML Feature** query (ML-2) `FD(X,Y) AND IND(X,W) AND FD(Z,W)` can identify the same feature dependencies, but with `X` and `Z` being stored in different relations – a join on `X = W` basically results in the pattern of ML-1.

Data Analytics (DA) Analyzing the metadata of a relational dataset can provide valuable insights into the stored entities' structure and implicit logic. While data analytics describes the process of exploring and interpreting (meta-)data, data profiling is needed to extract the metadata beforehand. With the **Common Cause** query (DA-1) `FD(X,Y) AND FD(X,Z)`, for example, we can find column combinations `Y` and `Z` that rely on the same cause `X`. A possible instance of this query are the FDs {*Time,Material*} → {*Cost*} and {*Time,Material*} → {*Value*}, as the production time and resource material spend on the production of jewelry define both its production costs and its value. With the **Common Effect** query (DA-2) `FD(X,Y) AND FD(Z,Y)`, on the contrary, we can find column combinations `X` and `Z` that both have an effect on `X`. For instance, we could find {*Revenue*} → {*Sales*} and {*StockOpening*, *StockClosing*} → {*Sales*} and learn that the number of sales can be deduced from both the achieved revenue and stock changes. The cause and effect queries can both be extended with an IND to link causes and effects (DA-3 and DA-4), respectively, across different relations.

Schema Matching (SM) Integrating datasets with different schemata requires a process that finds correspondences between the attributes of the schemata. This process is called schema matching. It can compare the schemata's attributes based on their labels, instance data, and/or structural properties [MBR01; RB01]. These structural properties include statistics, data types, simple aggregates, and also data dependency combinations. Data profiling and DPQL, in particular, can be used to extract the necessary metadata automatically. For example, the **Uniqueness Match** (SM-1) can use the query `IND(X,Y) AND UCC(X) AND UCC(Y) AND SPLIT(X,Y)` to find that `X` and `Y` are both candidate keys (`UCC`) in their respective relations (`SPLIT`) and, hence, structurally very similar; the inclusion dependency links the two column combinations `X` and `Y` and could be relaxed (e.g. to an overlap of 80%) when issued. Following the idea of similarity flooding [MGR02], structural information can also be used to allow similarities to rub off on semantically related attributes. With the **Dependent Match** (SM-2) query `IND(X,Y) AND FD(X,Z) AND FD(Y,W) AND SPLIT(X,Y)`, for instance, we find that both `Z` and `W` are dependent attributes of matching left-hand side column combinations `X` and `Y`; through this structural similarity they could inherit some of the similarity of `X` and `Y`. The **Determinant Match** uses the same intuition in SM-3, but lets the similarity of matching FD right-hand side attributes rub off on determinant attributes. Both queries are especially helpful if attribute labels and instance values differ greatly due to, for example, formatting, abbreviation or encoding differences. The **Included Match** (SM-4) and **Including Match** (SM-5) translate the same idea from FDs to INDs; the matched attributes `Z` and `W` may, then, even stem from other relations than `X` and `Y`.

Schema Analytics (SA) Schemata often exhibit interesting patterns, such as relational structures that form cliques, hubs, and authorities. A **3-Clique** (SA-1), for example, refers to three relations that all reference each other. The relations can be discovered with the DPQL query `IND(X,Y) AND IND(Y,Z) AND IND(Z,X) AND SPLIT(X,Y) AND SPLIT(Y,Z) AND SPLIT(Z,X)` with the `SPLIT`s forcing the column combinations `X`, `Y`, and `Z` to different relations. The **n-Clique** query (SA-2) generalizes SA-1 to cliques of larger size – we consider it as a 4-Clique in our evaluation. With inclusion dependencies and the `COALESCE(X,Z)` operator, which restricts `X` and `Z` to the same relation, further interesting schema patterns can be discovered. One of which is the **Hub** query (SA-3) `IND(X,Y) AND IND(Z,W) AND COALESCE(X,Z) AND SPLIT(X,Y) AND SPLIT(Z,W)`, which finds a relation that connects via `X` and `Z` to at least two other relations. Querying an **Authority** (SA-4) relation is a bit easier, because we expect other relations to reference the same key attribute(s) `Y`. An authority with two dependent relations can, hence, be found with the query `IND(X,Y) AND IND(Z,Y)`.

Further Applications While we named a handful of applications for DPQL queries that require holistic data profiling, many further applications exist. A particularly important application area is **data cleaning**, which requires (patterns of) *relaxed* dependencies to identify data errors, such as anomalies, inconsistencies, duplicates, missing values, and outliers [Ju25; KPN20]. For repairs, patterns such as DA-1 or DA-2 can be used [LKR20; Mi20]. In DPQL, data dependencies can be relaxed by adding the relaxation to the dependency definition to, for example, create a 95% valid IND $X \subseteq Y$ with `IND(X,Y,0.95)`. Additionally, for **integrity checking**, DPQL can be used to formulate specific constraints on a dataset and, in this way, ensure special consistency requirements.

3 Evaluation

In this section, we evaluate the queries in Tab. 1 on the TPCH [TP22] and AdventureWorks [Mi23] datasets. For each query, we use the pattern described in Tab. 1 as the `WHERE`-clause, specify all relations of the datasets in `CC`-terms in the `FROM`-clause, and report all resulting column combinations in the `SELECT`-clause. In the evaluation, we consider only those queries that are formed exclusively of INDs, FDs, and UCCs, because our current engine[3] [Se23] supports only these three types. We measure the DPQL queries' result sizes (i. e., sets of *dependency combinations*) and compare them to the result sizes of traditional profiling algorithms (i. e., sets of *primitive dependencies*); these sets contain all maximal INDs, minimal FDs, and/or minimal UCCs for the individual query patterns, which can be profiled with any existing exact dependency profiling algorithms [Ab18][4]. The measurements for this experiment are shown in Fig. 1. The y-axis shows the size of the DPQL result set as a percentage of the overall number of minimal/maximal primitive dependencies (UCCs, FDs, and/or INDs). Because we evaluate the queries on two datasets,

[3] https://github.com/SeegerM/Metaserve

[4] In our setup, we used BINDER [Pa15b], HyFD [PN16], and HyUCC [PN17a]

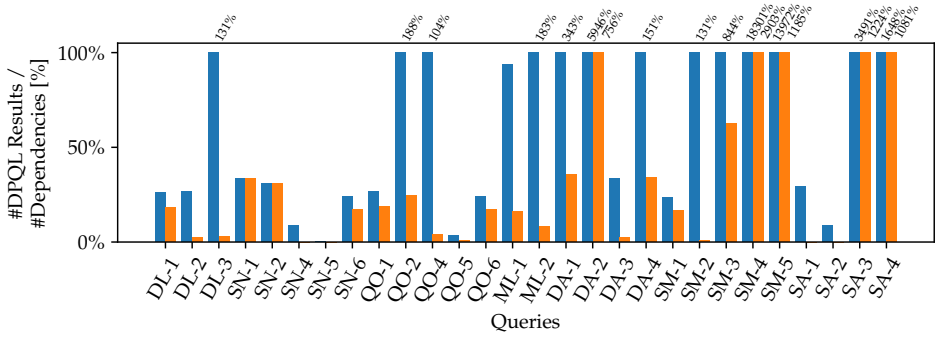

Fig. 1: Size of a DPQL result set relative to the size of the individually profiled (minimal/maximal) dependencies of a query pattern; blue bars are raw patterns and orange bars use additional filtering (i. e. **`NOT`** and **`CARDINALITY`**). Bars over 100% show the exact percentages as annotations above the bars.

every plotted percentage is actually the average of the two individual percentages. The blue bars represent the result sizes of the patterns as they are listed in Tab. 1; for the orange bars, we added some obvious, but optional additional filter criteria to the queries, such as that some FDs should not be UCCs (**`NOT UCC(X)`**) or that INDs should have a minimum coverage of 10% (**`CARDINALITY`**`(X) >= 0.1 * `**`CARDINALITY`**`(Y)`). If bars are at 100%, both results are equally large; if they are <100%, the DPQL result set is smaller; if they are >100% (indicated by the percentages above the bars) DPQL produces more combinations than all individual minimal/maximal dependency types combined. As shown in Fig. 1, about half of the DPQL result sets are smaller, which indicates a higher candidate pruning potential for their discovery. The other halve has slightly to clearly larger result sizes. This is because FDs (DL-3, DA-1 to DA-4 and SM-2 and SM-3) and INDs (SM-4, SM-5, and SA-1 to SA-4) often form combinations that generate many solutions of left and right dependencies. For example, the solution to DA-2 creates a result tuple for every combination of FDs `X` → `Y` and `Z` → `Y` that share the same `Y` value. Furthermore, many dependency combinations are formed of non-minimal/non-maximal dependencies and because these are more numerous than the minimal/maximal dependencies in the traditional result sets, more solutions arise despite DPQL's additional filtering potential. Because all these additional combinations are valid and potentially useful solutions, traditional profiling clearly does not meet the demand. However, the orange bars show that DPQL offers appropriate filtering tools (**`SPLIT`**, **`CARDINALITY`**, **`SIZE`** etc.) to refine large results further and make them smaller if needed.

Because the combination of different dependencies often causes an explosion in the number of results, we now compare in Fig. 2 the number of DPQL results with the full cross-product of all minimal/maximal dependencies, which are essentially all candidate dependency combinations that a data scientist has to generate and test if only traditionally profiled dependencies are given. The foreign-key discovery on AdventureWorks, for example, finds 5209 maximal INDs and 289 minimal UCCs resulting in 5209 × 289 = 1 505 401 potential

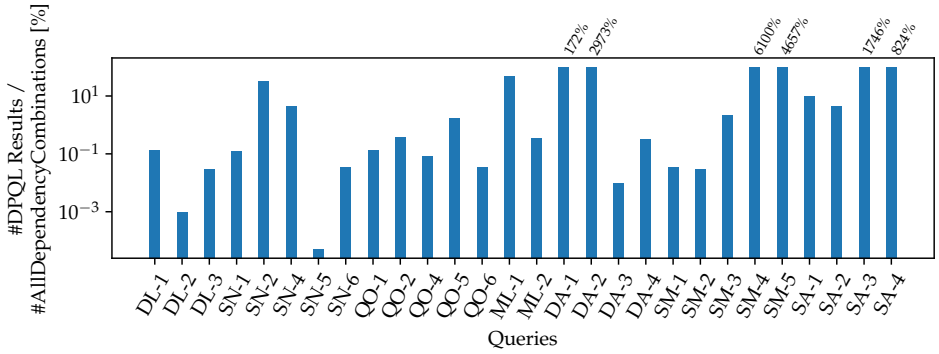

Fig. 2: Size of a DPQL result set relative to the size of all dependency combination candidates that can be derived directly from individual (minimal/maximal) dependencies; y-axis uses log scale. Bars over 100% show the exact percentages as annotations above the bars.

combinations; via DPQL, we find only 2538 dependency combinations, which contain all 76 real foreign key relationships. So in comparison to all traditionally implied dependency combinations (see Fig. 2), DPQL result sets are on avg. 90% (max. 99%) smaller, which demonstrates the increased potential for additional candidate pruning during profiling. Some DPQL results are still larger, which again shows that simple combinations of traditionally profiled dependencies without further specialization (or generalization) do not fit pattern semantics at all and sophisticated post-processing is needed to form valid result statements.

4 Summary

In this paper, we surveyed a variety of applications for DPQL queries from different application areas showing that DPQL serves to effectively combine unique column combinations, functional dependencies, inclusion dependencies, order dependencies, and matching dependencies for these applications. We formulated several concrete DPQL query patterns, evaluated them, and compared the result sizes to the result sizes of standard data profiling algorithms. The evaluation showed that DPQL query result sets are often much smaller, but sometimes also significantly larger than result sets of traditional profiling runs. No matter how one judges the size of result sets, the results clearly emphasize the discrepancy between traditional profiling results and DPQL profiling results, i.e., dependency patterns that applications actually need. This paper, in summary, demonstrates the usefulness of DPQL for different applications in producing exactly the required dependency combinations; the produced result sets are often even smaller than traditional result sets.

Acknowledgements. This work was supported by the Bundesministerium für Wirtschaft und Klimaschutz (BMWK.IIB5) Research Fund of Praezisions-LDS (03EE3061F). Additional funding was provided by the LOEWE initiative (Hesse, Germany) within the emergenCITY center [LOEWE/1/12/519/03/05.001(0016)/72]

References

[Ab18] Abedjan, Z.; Golab, L.; Naumann, F.; Papenbrock, T.: Data Profiling. In: Synthesis Lectures on Data Management (SLDM). 1st ed., Springer Cham, 2018, DOI: 10.1007/978-3-031-01865-7.

[BB79] Beeri, C.; Bernstein, P. A.: Computational Problems Related to the Design of Normal Form Relational Schemas. ACM Transactions on Database Systems (TODS) 4 (1), pp. 30–59, 1979, DOI: 10.1145/320064.320066.

[Bi20] Birnick, J.; Bläsius, T.; Friedrich, T.; Naumann, F.; Papenbrock, T.; Schirneck, M.: Hitting set enumeration with partial information for unique column combination discovery. Proceedings of the VLDB Endowment (PVLDB) 13 (12), pp. 2270–2283, 2020, DOI: 10.14778/3407790.3407824.

[Bl24] Bleifuß, T.; Papenbrock, T.; Bläsius, T.; Schirneck, M.; Naumann, F.: Discovering Functional Dependencies through Hitting Set Enumeration. Proceedings of the International Conference on Management of Data (SIGMOD) 2 (1), pp. 1–24, 2024, DOI: 10.1145/3639298.

[CFP82] Casanova, M. A.; Fagin, R.; Papadimitriou, C. H.: Inclusion dependencies and their interaction with functional dependencies. In: Proceedings of the Symposium on Principles of Database Systems (PODS). ACM New York, NY, USA, Los Angeles, USA, pp. 171–176, 1982, DOI: 10.1145/588111.588141.

[Ch19] Chen, H.; Jajodia, S.; Liu, J.; Park, N.; Sokolov, V.; Subrahmanian, V. S.: FakeTables: Using GANs to Generate Functional Dependency Preserving Tables with Bounded Real Data. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). International Joint Conferences on Artificial Intelligence Organization, pp. 2074–2080, 2019, DOI: 10.24963/ijcai.2019/287.

[Ch23] Chen, J.; Jin, Y.; Li, Y.; Tan, Z.; Yang, W.; Ma, S.: Effective and Efficient Lexicographical Order Dependency Discovery. IEEE Transactions on Knowledge and Data Engineering (TKDE) 35 (9), pp. 9700–9714, 2023, DOI: 10.1109/TKDE.2023.3248780.

[Ch98] Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. Pp. 34–43, 1998, DOI: 10.1145/275487.275492.

[Co70] Codd, E. F.: A relational model of data for large shared data banks. Communications of the ACM (CACM) 13 (6), pp. 377–387, 1970, DOI: 10.1145/362384.362685.

[Co71] Codd, E. F.: Further Normalization of the Data Base Relational Model. IBM Research Report, San Jose, California RJ909, 1971.

[Co74] Codd, E. F.: Recent Investigations in Relational Data Base Systems. In: Information Processing, Proceedings of the IFIP Congress. Pp. 1017–1021, 1974.

[DR02] Do, H.-H.; Rahm, E.: COMA – A System for flexible combination of schema matching approaches. In: Proceedings of the VLDB Endowment (PVLDB). VLDB Endowment, Hong Kong, China, pp. 610–621, 2002, DOI: 10.1016/B978-155860869-6/50060-3.

[Dü19] Dürsch, F.; Stebner, A.; Windheuser, F.; Fischer, M.; Friedrich, T.; Strelow, N.; Bleifuß, T.; Harmouch, H.; Jiang, L.; Papenbrock, T.; Naumann, F.: Inclusion Dependency Discovery: An Experimental Evaluation of Thirteen Algorithms. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). ACM New York, NY, USA, Beijing, China, pp. 219–228, 2019, DOI: 10.1145/3357384.3357916.

[Fa77] Fagin, R.: Multivalued dependencies and a new normal form for relational databases. ACM Transactions on Database Systems (TODS) 2 (3), pp. 262–278, 1977, ISSN: 0362-5915, DOI: 10.1145/320557.320571.

[Fe18] Fernandez, R. C.; Abedjan, Z.; Koko, F.; andSamuel Madden, G. Y.; Stonebraker, M.: AURUM: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE, Paris, France, 2018, DOI: 10.1109/ICDE.2018.00094.

[HL23] Hamdard, M. S.; Lodin, H.: Effect of feature selection on the accuracy of machine learning model. International Journal of Multidisciplinary Research and Analysis 6 (9), pp. 4460–4466, 2023, DOI: 10.47191/ijmra/v6-i9-66.

[HTD09] Hua, J.; Tembe, W. D.; Dougherty, E. R.: Performance of feature-selection methods in the classification of high-dimension data. Pattern Recognition 42 (3), pp. 409–424, 2009, DOI: 10.1016/j.patcog.2008.08.001.

[Hu04] Hua, J.; Xiong, Z.; Lowey, J.; Suh, E.; Dougherty, E. R.: Optimal number of features as a function of sample size for various classification rules. Bioinformatics 21 (8), pp. 1509–1515, 2004, DOI: 10.1093/bioinformatics/bti171.

[IC15] Ilyas, I. F.; Chu, X.: Trends in Cleaning Relational Data: Consistency and Deduplication. Foundations and Trends® in Databases 5 (4), pp. 281–393, 2015, DOI: 10.1561/1900000045.

[JN20] Jiang, L.; Naumann, F.: Holistic primary key and foreign key detection. Journal of Intelligent Information Systems (JIIS) 54, pp. 439–461, 2020, DOI: 10.1007/s10844-019-00562-z.

[Ju25] Jung, P.; Biessmann, F.; Neutatz, F.; Abedjan, Z.; Papenbrock, T.: Mimir: A Holistic Error Correction System (UNDER REVIEW). The VLDB Journal (VLDBJ) 1 (1), pp. 1–23, 2025.

[Ko22] Kossmann, J.; Lindner, D.; Naumann, F.; Papenbrock, T.: Workload-driven, lazy discovery of data dependencies for query optimization. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR). Vol. 12, CIDR Conference, Santa Cruz, USA, pp. 1–7, 2022.

[KPN20] Koumarelas, l.; Papenbrock, T.; Naumann, F.: MDedup: Duplicate Detection with Matching Dependencies. Proceedings of the VLDB Endowment (PVLDB) 13 (5), pp. 712–725, 2020, DOI: 10.14778/3377369.3377379.

[KPN22] Kossmann, J.; Papenbrock, T.; Naumann, F.: Data dependencies for query optimization: a survey. The VLDB Journal (VLDBJ) 31 (1), pp. 1–22, 2022, DOI: 10.1007/s00778-021-00676-3.

[KY83] Kambayashi, Y.; Yoshikawa, M.: Query processing utilizing dependencies and horizontal decomposition. In: Proceedings of the International Conference on Management of Data (SIGMOD). Pp. 55–67, 1983, DOI: 10.1145/971695.582205.

[Li24] Li, Y.; Li, R.; Tan, Z.; Yang, W.; Ma, S.: Efficient Set-Based Order Dependency Discovery with a Level-Wise Hybrid Strategy. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE, pp. 692–704, 2024, DOI: 10.1109/ICDE60146.2024.00059.

[LKR20] Livshits, E.; Kimelfeld, B.; Roy, S.: Computing optimal repairs for functional dependencies. ACM Transactions on Database Systems (TODS) 45 (1), pp. 1–46, 2020.

[LO78] Lucchesi, C. L.; Osborn, S. L.: Candidate keys for relations. Journal of Computer and System Sciences 17 (2), pp. 270–279, 1978, DOI: 10.1016/0022-0000(78)90009-0.

[LRN24] Lindner, D.; Ritter, D.; Naumann, F.: Enabling Data Dependency-based Query Optimization. arXiv preprint arXiv:2406.06886, 2024.

[LV00] Levene, M.; Vincent, M. W.: Justification for inclusion dependency normal form. IEEE Transactions on Knowledge and Data Engineering (TKDE) 12 (2), pp. 281–291, 2000, DOI: 10.1109/69.842267.

[MBR01] Madhavan, J.; Bernstein, P. A.; Rahm, E.: Generic schema matching with cupid. In: vldb. Vol. 1 2001. 2001, pp. 49–58, 2001.

[MGR02] Melnik, S.; Garcia-Molina, H.; Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE, pp. 117–128, 2002, DOI: 10.1109/ICDE.2002.994702.

[Mi20] Miao, D.; Cai, Z.; Li, J.; Gao, X.; Liu, X.: The computation of optimal subset repairs. Proceedings of the VLDB Endowment 13 (12), pp. 2061–2074, 2020.

[Mi23] Microsoft: AdventureWorks sample databases, 2023, URL: https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16.

[Pa00] Paulley, G. N.: Exploiting Functional Dependence in Query Optimization, tech. rep., University of Waterloo, 2000.

[Pa15a] 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. Proceedings of the VLDB Endowment (PVLDB) 8 (10), pp. 1082–1093, 2015, DOI: 10.14778/2794367.2794377.

[Pa15b] Papenbrock, T.; Kruse, S.; Quiané-Ruiz, J.-A.; Naumann, F.: Divide & conquer-based inclusion dependency discovery. Proceedings of the VLDB Endowment (PVLDB) 8 (7), pp. 774–785, 2015, DOI: 10.14778/2752939.2752946.

[PL10] Paulley, G. N.; Larson, P.-Å.: Exploiting uniqueness in query optimization. In: Proceedings of the International Conference on Data Engineering (ICDE). Pp. 127–145, 2010, DOI: 10.1109/ICDE.1994.283003.

[PN16] Papenbrock, T.; Naumann, F.: A Hybrid Approach to Functional Dependency Discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD). Pp. 821–833, 2016, DOI: 10.1145/2882903.2915203.

[PN17a] Papenbrock, T.; Naumann, F.: A Hybrid Approach for Efficient Unique Column Combination Discovery. In: Proceedings of the Conference Database Systems for Business, Technology and Web (BTW). Pp. 195–204, 2017, ISBN: 978-3-88579-659-6.

[PN17b] Papenbrock, T.; Naumann, F.: Data-driven Schema Normalization. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Vol. 17, Venice, Italy, pp. 342–353, 2017.

[RB01] Rahm, E.; Bernstein, P. A.: A survey of approaches to automatic schema matching. The VLDB Journal (VLDBJ) 10 (4), pp. 334–350, 2001, DOI: 10.1007/s007780100057.

[Ro09] Rostin, A.; Albrecht, O.; Bauckmann, J.; Naumann, F.; Leser, U.: A Machine Learning Approach to Foreign Key Discovery. In: Proceedings of the ACM Workshop on the Web and Databases (WebDB). Vol. 12, ACM New York, NY, USA, Rhode Island, USA, pp. 1–6, 2009, DOI: 10.14569/IJACSA.2021.0120480.

[Sa21] Salam, M. A.; Azar, A. T.; Elgendy, M. S.; Fouad, K. M.: The effect of different dimensionality reduction techniques on machine learning overfitting problem. Int. J. Adv. Comput. Sci. Appl 12 (4), pp. 641–655, 2021.

[SC23] Song, S.; Chen, L.: Integrity Constraints on Rich Data Types. In: Synthesis Lectures on Data Management (SLDM). Springer Nature, Cham, 2023, DOI: 10.1007/978-3-031-27177-9.

[Se23] Seeger, M.; Schmidl, S.; Vielhauer, A.; Papenbrock, T.: DPQL: The Data Profiling Query Language. Proceedings of the Conference Database Systems for Business, Technology and Web (BTW) 20, pp. 319–415, 2023, DOI: 10.18420/BTW2023-19.

[SGG12] Szlichta, J.; Godfrey, P.; Gryz, J.: Fundamentals of order dependencies. Proceedings of the VLDB Endowment (PVLDB) 5 (11), pp. 1220–1231, 2012, DOI: 10.14778/2350229.2350241.

[So20] Song, S.; Gao, F.; Huang, R.; Wang, C.: Data dependencies extended for variety and veracity: A family tree. IEEE Transactions on Knowledge and Data Engineering (TKDE) 34 (10), pp. 4717–4736, 2020, DOI: 10.1109/TKDE.2020.3046443.

[SP22] Schmidl, S.; Papenbrock, T.: Efficient Distributed Discovery of Bidirectional Order Dependencies. The VLDB Journal (VLDBJ) 31 (1), pp. 49–74, 2022, DOI: 10.1007/s00778-021-00683-4.

[TP22] TPC: TPC-H Benchmark, 2022, URL: http://www.tpc.org/tpch/.

[VA18] Visengeriyeva, L.; Abedjan, Z.: Metadata-Driven Error Detection. In: Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM). Vol. 30, ACM New York, NY, USA, Bolzano, Italy, pp. 1–12, 2018, DOI: 10.1145/3221269.3223028.

[Zh10] Zhang, M.; Hadjieleftheriou, M.; Ooi, B. C.; Procopiuc, C. M.; Srivastava, D.: On Multi-column Foreign Key Discovery. Proceedings of the VLDB Endowment (PVLDB) 3 (1-2), pp. 805–814, 2010, DOI: 10.14778/1920841.1920944.