Making DBMSes Dependency-Aware

George Chernishev, JetBrains Research, Information Syst. Eng. Lab., chernishev@gmail.com

Recently, discovery of functional (both exact and approximate), conditional, inclusion, and other types of dependencies has experienced a surge of interest [2–4]. Among others, the Metanome project [1] offers a plethora of high performance algorithms for mining all kinds of dependencies.

Up until now, mined dependencies have existed as relatively passive database objects. Functional dependencies (FDs) have been used for data cleaning and analysis, as well as in several proposals for query optimization. However, even they still are largely external objects and DBMSes are mostly unaware of them.

Our idea is to "attach" dependencies to their respective table and allow the user to query them and to use them for querying data, thus making the DBMS dependency-aware. Effectively, this will include:

① an ability to query dependencies contained in a table;

② an ability to use filtering conditions using dependencies while querying the data itself (in the SELECT clause);

③ an ability to pre-mine and store dependencies.

The goal is to provide the user with in-database analysis tools that employ various dependency concepts. It will allow users to discover regularities in data, that in turn will lead to generation of hypotheses and domain-specific conclusions. Such analysis would be useful for business and scientific applications (e.g., astronomy, bioinformatics) where data is represented via wide tables.

Note that here we do not specify which types of dependencies should be used. Intuitively, one may say that exact and approximate FDs should be the first candidates for implementation. In reality, this depends on the domain area. For example, the biologists we have interviewed were more interested in differential and conditional dependencies.

① allows to easily navigate a collection of mined dependencies. Despite the fact that usually only minimal, non-trivial dependencies are of interest to users, it is still a problem to present them all for analysis: a table featuring several dozens of attributes may have millions of dependencies. State-of-the-art approaches output FDs either as a plain list or using simple visualization techniques [1], which are not suitable even for medium-sized tables. Therefore, a special language which will allow to declaratively query a collection of FDs is needed.

Some examples of user queries: "for a given table, output all FDs that involve a given attribute", "for a given table and a given attribute, output all FDs that functionally determine it".

② allows to perform in-depth analysis of data while employing various dependency-related predicates. For example, an analyst may be interested in such queries as: "find all rows which prevent a given functional dependency from holding" or "project all attributes that functionally determine a specified attribute".

③ is motivated by the fact that dependency discovery even for small tables takes hours even on modern server-class hardware. Therefore, an online approach is not viable. Instead, dependencies should be pre-mined in advance of querying and stored in a special table. This table will serve as an intermediate layer between mining and querying. It will be controlled by the user who specifies its population, i.e. what will be mined. Moreover, in order to make the proposed analytics viable, it allows the user to restrict the length of the desired dependency and explicitly guide the discovery process (e.g. by selecting participating attributes). This will allow to reduce run times of discovery algorithms.

It is worth to move such analysis inside a database due to the following: **1)** SQL and RDBMes are immensely powerful and convenient tools to query and explore data. These qualities stem from their declarative nature and the presence of a query optimizer, which result in unparalleled flexibility. Such capabilities are required during FD-related data exploration. For example, one may want to re-check the presence of a FD on a subset of a table. Filtering it outside is inconvenient and costly. **2)** FDs are inseparable from data in a sense that they should be stored together with the data they belong to. Moving them outside will require synchronization in case of changes in data. This can be prohibitively expensive in some cases. **3)** Finally, there is a trend for in-database analytic processing. Note that a similar integration happened to XML processing, temporal extensions, and is currently happening to ML.

Designing this system will require: 1) for ①, extending SQL with clauses that support selecting a dependency out of a dependency collection, 2) for ②, designing a set of dependency-related predicates (which will be dependency-specific) and modifications of the query engine, 3) for ③, integrating existing discovery algorithms into DBMSes, which will most likely involve the partition [2] intersection approach. Therefore, column-stores are the preferred choice for ensuring maximum performance.

[1] T. Papenbrock et al. 2015. Data Profiling with Metanome. Proc. VLDB Endow. 8, 12 (Aug. 2015), 1860–1863.
[2] T. Papenbrock and F. Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery (SIGMOD'16).
[3] P. Schirmer et al. 2019. DynFD: Functional Dependency Discovery in Dynamic Datasets (EDBT'19).
[4] L. Caruccio et al. 2017. Evolutionary Mining of Relaxed Dependencies from Big Data Collections (WIMS'17).
[5] L. Caruccio et al. 2016. Relaxed functional dependencies — a survey of approaches. IEEE TKDE, 28(1).

9th Biennial Conference on Innovative Data Systems Research (CIDR '19) January 13-16, 2019 , Asilomar, California, USA.