Technical Perspective: Attacking the Problem of Consistent Query Answering

Wang-Chiew Tan
University of California, Santa Cruz
tan@cs.ucsc.edu

Inconsistent data refers to data that do not adhere to one or more constraints. The term *constraints* refers to conditions that need to be imposed on the data. Constraints often arise from organizational requirements or business logic, such as the requirement that every employee in the database must be uniquely identified by the employee id, or every employee must work on some project, or the expenses cannot exceed the credit limit, or even a desired designated format for storing phone numbers. The need to manage *inconsistent data* arises in many settings. Quite typically, when one integrates data from different sources, the integrated data can be inconsistent data even when the data sources may be individually consistent. Another scenario where inconsistency in data can arise is when data and/or schema evolves, for example, through the addition or removal of data, changes in schema, or knowledge of new constraints.

There are generally two themes of research centered around the management of inconsistent data. One theme of research aims to make the data *clean* before the data is allowed to be used or queried. In other words, data is modified so that it becomes consistent prior to the execution of queries over the data. The data cleaning process is typically algorithmic, and sometimes heuristic, so that data can be manipulated, somehow, into a final consistent state. While this approach produces one final clean dataset that one can work with, it is generally difficult to understand how the consistent state of the data was arrived at. In contrast to data cleaning, the other theme of research adopts a "lazy" perspective towards the management of inconsistent data. In this line of research, the inconsistent data is left unmodified and work to determine what are the right answers is done only when queries are to be executed over the inconsistent data. In other words, the management of inconsistencies only occurs at query time. To compute the answer of the query, one considers all possible ways to *repair* the inconsistent database and the intersection of the results of the query on each repair forms the answer to the query. Since these are answers that appear in the result of the query applied to every repair, they are called the *consistent answer* to the query [1]. A notion of repair that has been widely considered in the past is that of a consistent database instance that differs from the original inconsistent database in a "minimal" way. If the only constraints were key constraints, this amounts to minimally removing tuples from the inconsistent database so that the key constraints will no longer be violated.

The problem of computing the consistent query answer of a query over an inconsistent database (CQA for short) has received significant attention in the past several years. This problem is known to be coNP-complete in general for the class of conjunctive queries under primary key constraints [2]. However, for a number of years, it was unclear whether one can provide exact conditions to determine, even for the special class of self-join-free conjunctive queries, the exact complexity of the query. There was a flurry of research activities on this problem in the past decade or so and finally, the problem is resolved in [3], where the authors showed a trichotomy result; for any self-join-free boolean conjunctive query, it can be decided with an effective procedure whether or not the CQA problem is in FO, P, or coNP-complete.

There are immediate practical implications of this result. Existing implementations of systems for CQA tend to adopt an (overly) expressive and hence computationally expensive engine for computing the answer, or identify special classes of queries for which CQA can be computed by means of SQL queries. The latter allows one to leverage highly optimized relational database management systems and tends to run fast. With this result at hand, it is now possible to determine and delegate the computation of CQA to the right engine; pushing computations to the RDBMS whenever possible, and relying on more algorithmic or expressive engines otherwise.

Now, if you are curious about how they "attacked" the CQA problem, read on.

References

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[2] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. In *Information and Computation*, 197(1-2):90–121, 2005.

[3] P. Koutris and J. Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS)*, pages 17–29, 2015.