Provenance in Databases: Principles and Applications

Pierre Senellart[1,2,3][0000−0002−7909−5369]

[1] DI ENS, ENS, CNRS, PSL University, Paris, France
[2] Inria, Paris, France
[3] LTCI, Télécom Paris, IP Paris, Paris, France
pierre@senellart.com
http://pierre.senellart.com/

Abstract. Data provenance is extra information computed during query evaluation over databases, which provides additional context about query results. Several formal frameworks for data provenance have been proposed, in particular based on provenance semirings. The provenance of a query can be computed in these frameworks for a variety of query languages. Provenance has applications in various settings, such as probabilistic databases, view maintenance, or explanation of query results. Though the theory of provenance semirings has mostly been developed in the setting of relational databases, it can also apply to other data representations, such as XML, graph, and triple-store databases.

Keywords: Provenance· Databases

1 Introduction

This short paper provides a very high-level overview of the principles and applications of provenance in databases. A more in-depth but still accessible presentation of the same concepts can be found in [21]; we also refer the reader to the other references listed in this paper.

We first briefly define data provenance in Section 2, then highlight a few example applications in Section 3 before discussing provenance over databases that are not in the classical relational setting in Section 4.

2 Provenance

The main task in data management is *query evaluation*: given a database D (in some structured form) and a query q (from some class), compute the result of the query over the database, $q(D)$. In the most commonly used setting of relational databases [1], for example, a database is a collection of named tables, a query can be expressed in the SQL query language, and the result of a query is a table.

However, in a number of applications (see examples in Section 3), knowing the query result is not enough: it is also useful to obtain extra information

about this result, where it comes from, or how it has been computed. We call this extra information *data provenance* [5,8]. *Provenance management* deals with the computation of data provenance.

Data provenance can take multiple forms, depending on what kinds of information is required. A good and simple example of this is *Boolean provenance*, a notion introduced in [19] under a different terminology. Let X be a set of Boolean variables (variables that can be set to the values 0 or 1). We assume that every valuation ν of the variables of X, when applied to the database D, defines a new database $\nu(D)$. For example, if D is a relational database, every tuple of D can be associated with a different variable of X, and then $\nu(D)$ is simply the subdatabase of D formed only of tuples whose associated variable is set to 1 by ν. Then, by definition, the *provenance* of an element t in the query result $q(D)$ (e.g., in the relational setting, a tuple $t \in q(D)$) is the function from valuations of X to $\{0,1\}$:

$$\nu \mapsto \begin{cases} 1 \text{ if } t \in q(\nu(D)) \\ 0 \text{ otherwise.} \end{cases}$$

Boolean provenance is useful because the Boolean provenance of t in $q(D)$ is sufficient to determine the presence of t in any database of the form $\nu(D)$. In other words, if the Boolean provenance can be efficiently computed, it is possible to answer many kinds of hypothetical questions about what the output of the query q is over other databases than the database D.

Boolean provenance is special in that it can be defined quite abstractly, independently of a query language or even a precise data model. This definition, however, does not yield an efficient computation. A seminal paper on data provenance [17] has shown that, if we restrict the data model to relational databases and the query language to the *positive relational algebra* (the SELECT-FROM-WHERE core of SQL), Boolean provenance is simply a particular case of *semiring provenance*, and all forms of semiring provenances can be computed efficiently under the same restrictions. A semiring is an algebraic structure with two operators, $\oplus$ and $\otimes$, verifying some axioms; when semirings are used for provenance, the $\oplus$ operator corresponds to different possible ways of producing a given result (e.g., with union and duplicate elimination in the relational algebra), while $\otimes$ is used to indicate different information that need to be combined to produce a result (e.g., with joins and cross products). Semiring provenance, which is parameterized by an arbitrary commutative semiring – Boolean provenance corresponds to a parameterization by the semiring of Boolean functions –, captures most existing provenance formalisms, and yields multitude applications. See [17,21] for precise definitions.

3 Example Applications

We now list a few important applications of different forms of data provenance. The list is by no means restrictive, see, e.g., [21] for other examples.

Probabilistic databases. *Probabilistic databases* [23] are probability distributions over regular databases, these distributions being represented in some compact format. The central question in probabilistic databases is *probabilistic query evaluation*, namely computing the probability that a query is satisfied over a database. It turns out [18,23] that this problem can be solved using Boolean provenance: first, assign Boolean variables to the input database, and assign probabilities to these variables in a way consistent with the probability distribution; second, compute the Boolean provenance of the query; third, compute the probability that the Boolean provenance, seen as a Boolean function, evaluates to 1. This last part is intractable in general (#P-hard) but is amenable to techniques from the field of *knowledge compilation* [10,22].

View maintenance and view update. In databases, materialized views are stored representations of the result of a given query. If the original database is updated (e.g., through the deletion of some tuples), the materialized view needs to be maintained so as to reflect the new output of the query, hopefully without fully recomputing it; this is the *view maintenance* problem. Conversely, it should be possible (at least in simple situations) to issue an update (e.g., a deletion) over the content of the materialized view, and that this update be propagated to the original database; this is the *view update* problem. Both these problems can be solved using data provenance: View maintenance under deletions can be solved by maintaining the Boolean provenance of the view, and deleting tuples whose provenance evaluates to 0 once the variables associated to original deleted tuples are set to 0. View update under deletions can be solved using why-provenance [6], a form of semiring provenance.

Explanation of query results. Different forms of provenance can also be used to present a user with explanation of query results: *where-provenance* [5] can explain where a particular data value in the output comes from; *why-provenance* [5] which data inputs have been combined to produce a query result; *how-provenance* [17] how the entire result was constructed; *why-not provenance* [7] why a particular result was not produced. Though why- and how- provenance can be computed in the framework of semiring provenance, where- and why-not provenance require different techniques.

Provenance Management Systems In order to support such applications, a number of provenance management systems have been designed. We restrict the discussion here to general-purpose provenance management in database systems, and not in other settings, such as scientific workflows [11]. Perm [16] modifies the internals of a now-obsolete PostgreSQL relational database management system to add support for computation of provenance. This design, unfortunately, had made it hard to maintain the system and to deploy it in modern environments. GProM [4] and ProvSQL [22] are two more recent provenance management systems which address this issue in two different ways: GProM is implemented as a middleware between the user and a database system, queries being rewritten on the fly to compute provenance annotations; ProvSQL is implemented as a

lightweight add-on to PostgreSQL, which can be deployed on an existing PostgreSQL installation. GProM and ProvSQL both support provenance computation in various provenance semirings; ProvSQL also is a probabilistic database system, computing probabilities from the Boolean provenance. See the discussion in [22] for a comparison of the main features of GProM and ProvSQL.

4 Beyond Relational Provenance

Most research on provenance (and in particular on semiring provenance) has been carried out in the common setting of relational databases for the simple query language of the positive relational algebra. Extensions to richer query languages, and to different data models, are possible, though sometimes with different approaches.

Non-monotone queries. Semiring provenance can only capture the provenance of monotone queries, such as those of the positive relational algebra. Moving to non-monotone queries and the full relational algebra requires considering *semirings with monus* [2,14], where the monus $\ominus$ operator is used to represent negative information.

Aggregation. In order to capture the provenance of aggregation operators (such as sum or count), it is necessary to move from semirings to semimodules over the scalars that are the aggregation values [3].

Recursive queries. To add support for query languages involving recursion (such as Datalog), it is necessary to add constraints on to which semirings are considered: depending on these constraints (e.g., ω-continuity [17], absorptivity [12], existence of a * operator [20]), different algorithms can be used to compute the provenance.

XML databases. XML databases organize information in a hierarchical, tree-like manner. Queries over XML databases typically resemble tree patterns to be matched over the tree database. Semiring provenance concepts can be extended to this setting in a quite straightforward manner [13].

Graph databases. In graph databases, data is represented as a labeled, annotated graph, and queries make it possible to ask for the existence of a path between two nodes with constraints on its labels. Graph queries are inherently recursive, and require similar techniques as to support Datalog queries over relational databases [20].

Triple stores. Triple stores model information using the Semantic Web standard of subject–predicate–object triples. Queries, for example expressed in the standard SPARQL query language, represent complex patterns of triples. Negation is an important feature of Semantic Web languages, so semirings with monus are also deployed in this setting [9]; these semirings with monus must also verify additional axioms imposed by the semantics of SPARQL [15].

5 Outlook

The principles of provenance management in databases are now well-understood. The framework of provenance semirings, in particular, has revealed to be very

fruitful. It also lends itself to a number of extensions beyond the positive relational algebra, as discussed in Section 4; some of these extensions are not fully fleshed out, however, and still require more work. Some other areas are in need of more research: for instance, on how updates in databases should interact with provenance annotations; or on how to combine provenance computation with efficient query processing. However, there are now enough foundations to build and optimize concrete provenance management systems (starting with the existing software, in particular, GProM and ProvSQL), and to apply them to real-world use cases.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Amer, K.: Equationally complete classes of commutative monoids with monus. Algebra Universalis **18**(1) (1984)
3. Amsterdamer, Y., Deutch, D., Tannen, V.: Provenance for aggregate queries. In: PODS (2011)
4. Arab, B.S., Feng, S., Glavic, B., Lee, S., Niu, X., Zeng, Q.: GProM - A swiss army knife for your provenance needs. IEEE Data Eng. Bull. **41**(1), 51–62 (2018)
5. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: ICDT (2001)
6. Buneman, P., Khanna, S., Tan, W.C.: On propagation of deletions and annotations through views. In: PODS (2002)
7. Chapman, A., Jagadish, H.V.: Why not? In: SIGMOD (2009)
8. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Foundations and Trends in Databases **1**(4) (2009)
9. Damásio, C.V., Analyti, A., Antoniou, G.: Provenance for SPARQL queries. In: ISWC (2012)
10. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artificial Intelligence Research **17**(1) (2002)
11. Davidson, S.B., Boulakia, S.C., Eyal, A., Ludäscher, B., McPhillips, T.M., Bowers, S., Anand, M.K., Freire, J.: Provenance in scientific workflow systems. IEEE Data Eng. Bull. **30**(4), 44–50 (2007)
12. Deutch, D., Milo, T., Roy, S., Tannen, V.: Circuits for Datalog provenance. In: ICDT (2014)
13. Foster, J.N., Green, T.J., Tannen, V.: Annotated XML: queries and provenance. In: PODS (2008)
14. Geerts, F., Poggi, A.: On database query languages for K-relations. J. Applied Logic **8**(2) (2010)
15. Geerts, F., Unger, T., Karvounarakis, G., Fundulaki, I., Christophides, V.: Algebraic structures for capturing the provenance of SPARQL queries. J. ACM **63**(1) (2016)
16. Glavic, B., Alonso, G.: Perm: Processing provenance and data on the same data model through query rewriting. In: ICDE. pp. 174–185 (2009)
17. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS (2007)
18. Green, T.J., Tannen, V.: Models for incomplete and probabilistic information. IEEE Data Eng. Bull. **29**(1) (2006)

19. Imielinski, T., Lipski, Jr., W.: Incomplete information in relational databases. J. ACM **31**(4) (1984)
20. Ramusat, Y., Maniu, S., Senellart, P.: Semiring provenance over graph databases. In: TaPP (2018)
21. Senellart, P.: Provenance and probabilities in relational databases: From theory to practice. SIGMOD Record **46**(4) (2017)
22. Senellart, P., Jachiet, L., Maniu, S., Ramusat, Y.: ProvSQL: provenance and probability management in PostgreSQL. PVLDB **11**(12) (2018)
23. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)