Research Problems in Data Provenance

Wang-Chiew Tan*
University of California, Santa Cruz
Email: wctan@cs.ucsc.edu

Abstract

The problem of tracing the provenance (also known as lineage) of data is an ubiquitous problem that is frequently encountered in databases that are the result of many transformation steps. Scientific databases and data warehouses are some examples of such databases. However, contributions from the database research community towards this problem have been somewhat limited. In this paper, we motivate the problem of supporting data provenance in scientific database applications and provide some background on previous research. We also briefly describe the DBNotes prototype developed at UC Santa Cruz that can be used to "eagerly" trace the provenance and flow of relational data and describe some directions for further research.

1 Introduction

The word "provenance" means "origin" or "source" and is often used in association with a piece of art or literature. Indeed, Merriam-Webster [19] additionally defines provenance as "the history of ownership of a valued object or work of art or literature". Knowledge on the history of ownerships of a work of art is of great importance and is evidenced by the number of provenance projects that we can find today, including the Provenance Research Project at the Metropolitan Museum [23] and at the Harvard University Art Museums [21]. According to [21], provenance "can help to determine the authenticity of a work, to establish the historical importance of a work by suggesting other artists who might have seen and been influenced by it, and to determine the legitimacy of current ownership". The motivation for understanding the provenance of works of art is also also applicable to data we see on the Web. With the proliferation of data on the Web, questions such as "Where did this data come from?", "Who else is using this data?", and "Why is this piece of data here?" are becoming increasingly common.

The field of molecular biology has some 500 databases [14] but only a handful of these are source databases in the sense that they receive experimental data directly. In fact, most databases are *curated* from other databases, i.e., they consists of data integrated from several other databases, often with extensive manual input and cleansing performed by the maintainers of the database. In such a scenario, it is natural for a user to ask if a piece of data in the curated database is added by the maintainers or derived from other source database in order to help determine the quality of information that she is receiving. Despite the importance of understanding the provenance of data, there has been relatively few foundational techniques and tools that have been developed to help one trace the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Supported in part by an NSF CAREER Award IIS-0347065 and NSF grant IIS-0430994.

provenance of data in scientific databases. Data can be physically moved from one database to another as a result of executing a query, a program or a manual transformation or virtually moved from one database to another as a result of high-level descriptions of the relationships that are specified between data sources. Unless there is a concerted effort to explicitly maintain provenance as data is moved around, such information is usually lost in a database transformation process. Also, the situation is often complicated by the fact that a source database may be evolving in both its content and structure over time. A complete description of data provenance in this scenario would involve tracing a piece of data to its correct source version and locating that piece of data in other versions, if any. This means that every version of the source database that ever existed must somehow be retained and the relationships between different versions, which may differ in both structure and content, must either be derivable or documented.

It should be noted that this paper is devoted to the discussion of the provenance of data, as opposed to the provenance of a data product even though both types of provenance are equally important [3, 16]. The latter deals with how a data product is derived. For example, the provenance of the result from a scientific experiment may include the workflow steps, software and the version of software, software parameters, the operating system, and raw data that are used in the experiment to arrive at the result.

In the next section, we provide some background on existing work related to data provenance and in Section 3, we describe a few applications where techniques for tracing data provenance are required. In Section 4, we briefly describe the DBNotes prototype developed at UC Santa Cruz and in Section 5, we describe some directions for further research.

2 Background

Existing techniques for tracing data provenance can be roughly classified under two approaches: "lazy" or "eager". Techniques for archiving data can generally be classified under two approaches: sequence-of-delta or timestamping.

2.1 The Lazy vs. Eager Approach to Tracing Data Provenance

The lazy approach computes the provenance of data only when needed. Existing work based on this approach generally assumes that the transformation process between the source database and the target database is available and is given as a query Q. The query Q and the output data d, whose provenance is of interest, are analyzed when needed and a new query is generated based on the analysis. When the new query is evaluated against the source database, it computes the provenance of d which are combinations of source tuples that together with the query Q, derive d. Existing techniques that uses this approach include [5, 11, 12, 31]. The problem of computing data provenance in the relational model was first considered in [31]. Provenance is traced at the attribute level and can be combined to obtain the provenance of data at a higher granularity (e.g., a tuple). The function that computes the output is "inverted" but this inverted function is a weak inverse in general; in addition to all the right answers returned, it may also return some wrong answers. A separate verification phase is used to remove the wrong answers. In [12], provenance is computed by analyzing queries in the relational algebra framework. A new query, which computes the provenance when it is applied to the source databases, is automatically generated from the analysis. Techniques for computing data provenance under general data warehouse transformations were proposed in [11]. In [5], an algorithm that can compute the provenance of a piece of data in a hierarchical deterministic data model in the result of a query was given. In addition, two kinds of provenance, why and where-provenance, were distinguished and characterized under this model. Intuitively, the *why-provenance* of a piece of data d in the result consists of sets of minimal pieces of source data such that each set, together with the query, is sufficient to reconstruct d in the output. On the other hand, the *where-provenance* of a piece of data d in the result is a more refined notion that tells us the exact pieces of source data where d is copied from.

Another approach to compute data provenance is to do so eagerly by carrying the provenance of data along as data is transformed [1, 2, 6, 17, 26, 29]. This approach has been given several names such as metadata support, source tagging, the attribution approach, or annotations. An advantage of using an eager method to compute data provenance is that the source databases need not be probed since the provenance can be fully determined by looking at the annotations associated with a piece of output data. A disadvantage is that the eager approach incurs additional overhead to compute the output and store annotations in the output. In Section 4, we shall describe the DBNotes prototype developed at UC Santa Cruz which allows one to eagerly propagate provenance along as data is transformed.

2.2 The Sequence-of-Delta vs. Timestamping Approach to Archiving

Many approaches for archiving data use the sequence-of-delta approach which stores a reference version (i.e., the first version or last version) and a sequence of forward or backward deltas between successive versions [9, 13, 18, 27]. For example, given a sequence of database versions V_1, V_2, ..., V_k, the sequence-of-delta approach may store V_1, together with Δ_{12} and Δ_{23}, ..., $\Delta_{k-1,k}$ where Δ_{ij} is the difference from version i to version j. Version V_1 together with the sequence of deltas Δ_{12}, ..., $\Delta_{i-1,i}$, is sufficient to reconstruct version i (and versions 1 through $i-1$ for that matter). Ideally, each delta is the smallest possible difference between two versions. A delta is typically computed with differencing techniques whose goal is to minimize the size of the delta or compute a reasonably small delta efficiently. A significant number of efficient algorithms for computing a delta, depending on the type of the file and application, have been proposed [7, 8, 10, 20, 22, 25, 32, 33].

In the timestamping approach, a "big" repository stores every version and timestamps are used to mark the existence of data elements at various times. For example, the source code control system (SCCS) [24] uses this technique. Given two versions to be archived, the difference (or correspondence of lines of text) between two versions is first computed. The difference is then used to timestamp lines of text in the repository to denote the addition or deletion of various lines of text at different times. The sequence-of-delta and timestamping approaches differ only in their change representation: the former technique describes the difference between two versions around time (through a delta) while the latter technique describes the difference around data (through timestamps).

These approaches which are based on diff often fail to capture the underlying semantics of data. As an example, if an object has changed some of its attribute values over time, a diff algorithm might erroneously report the change as that object exchanging its identity with another object [4]. This anomaly does not happen, however, if we only retrieve entire versions from the repository. It matters only when we are interested in retrieving changes pertaining to an object in the database. In [4], an archiving technique that avoids this anomaly by exploiting hierarchical key constraints has been developed.

3 Application Scenarios

In what follows, we provide some example scenarios where techniques for tracing the provenance of data and archiving are applicable.

Gauging the Trustworthiness of Data In an environment where data is repeatedly copied or edited on its journey from a source database to a new database, users are often interested to know the provenance of data in order to gauge the trustworthiness of data. Many scientific databases or biological databases, for example, fall into this scenario. These databases are often curated. For a user of such databases, it is crucial to understand whether the information she encounters in the database is added by the maintainers of that database, or derived from other databases for she may trust certain sources of information more than the others. This can also help

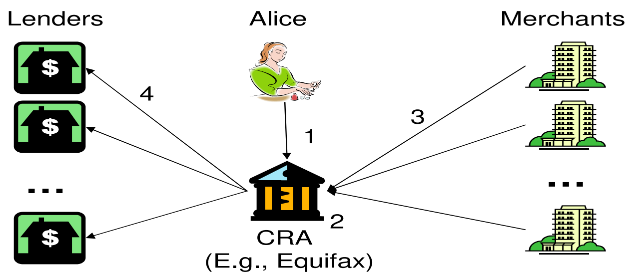

Figure 1: A complete example with the credit bureau.

determine the sources of error, if any, and gives a better understanding on the overall quality of the curated database.

Sharing Knowledge via Annotations There is usually a group of users associated with each database. It is common for these users to find errors, inconsistencies in the data, or discover additional information about the data and want to share their knowledge and experiences with other users. Since the database schema is often fixed and proprietary, one cannot simply modify the database to insert additional information that a user may have. A common practice nowadays is for the user to post her findings in a mailing list or newsgroup relevant for the community and hopefully, the changes will be reflected in the subsequent version of the database. However, unless every user makes a habit of reading and recording such announcements, it is likely that some valuable information will be lost in the long run.

Ideally, to share knowledge about a piece of data among users of the same community, one should be able to overlay the annotations on that piece of data and these annotations would automatically be reflected to other users who are looking at the same piece of data. The structure of annotations is not constrained by the schema of the database, if it exists. With a collaborative tool that allows annotations to be added and reviewed by many independent parties, it is likely that the quality of data in the database will be higher in the long run.

Verifying Data Recently, it is common in scientific literature to use references to Web pages to substantiate the claims made in the literature. The credibility of these claims therefore relies on the contents of the Web pages that have been cited. It happens frequently, however, that the contents of these Web pages change over time or become unavailable and hence, the claims are no longer verifiable. For scientific databases, research findings are often based on data of a particular version and it is therefore crucial to keep every version of the database that ever existed. The same reason applies to many other databases, such as financial databases or stock inventory databases, where a financial report or market-trend analysis may have been based on a particular version of the database and we would like these reports and analysis to remain verifiable.

3.1 Putting it all together: A Complete System for Tracing Data Provenance

We next describe a credit bureau example that would require one to apply techniques for tracing data provenance, annotation propagation and archiving in a single business process. See Figure 1.

1. A credit bureau[1] e.g., Equifax, receives a letter from Alice claiming an inaccuracy in her credit history report, dated two months ago.

2. Equifax retrieves Alice's record from their archive in order to verify her claim. This step involves tracing for Alice's two month old record in the archive and verifying her claim against the record in the database.

[1] Credit bureaus, also known as credit reporting agencies (CRAs), collect information from our creditors (e.g., merchants, landlords, etc.) about our credit habits and then sell this information to businesses (e.g., lenders) so they can evaluate our application for credit.

At the same time, Equifax also traces the record in the archive forward in time in order to verify that Alice's record has not been updated with the correct information since.

3. After having verified that Alice's record is indeed as claimed and has not been corrected, Equifax then checks for the provenance of the disputed entry in which it attempts to identify the merchant responsible for reporting the disputed entry. Equifax found that the merchant responsible for reporting the disputed entry has a known history of inaccurate reporting.

4. A formal investigation is then initiated by Equifax to investigate the dispute. Meanwhile, Equifax annotates the disputed entry in Alice's record with Alice's written statement of dispute so that potential lenders who are reviewing Alice's credit history are aware of the dispute.

The above example shows that in order for Equifax to smoothly handle the dispute from Alice, it requires an archival database system that has the capability of reporting the history of a record (in this case, a trace on Alice's record was made as described in Step 2), tracing the provenance of a disputed entry (Step 3) and propagating annotations of a record through views to the lenders (Step 4).

4 The DBNotes Project at UC Santa Cruz

DBNotes is an annotation management system developed at UC Santa Cruz that currently works on top of a relational database system. In DBNotes, every value can be associated with zero or more annotations. A *value* refers to an attribute value of a tuple in some relation. Annotations are automatically propagated along with the values as they are being transformed through a query. Currently, DBNotes supports the propagation of annotations through an extension of a fragment of SQL, called pSQL. In its default behavior, DBNotes propagate annotations based on where data is *copied* from. As a consequence, if every column of every tuple in a database is annotated with its address, the provenance of data is propagated along as data is transformed. An example is shown below. Suppose we have the following tables SWISS-PROT and GENBANK.

SWISS-PROT

ID	Desc
q229 $\{a_1\}$	CC $\{a_2\}$
q939 $\{a_3\}$	ED $\{a_4\}$

GENBANK

ID	Desc
g231 $\{a_5\}$	AB $\{a_6\}$
g756 $\{a_7\}$	CC $\{a_8\}$

Q_1:

```
select distinct Desc
from SWISSPROT
propagate default
union
select distinct Desc
from GENBANK
propagate default
```

Result of Q_1:

Desc
AB $\{a_6\}$
CC $\{a_2, a_8\}$
ED $\{a_4\}$

Each value has a distinct address and is denoted by the annotation "a_i", $1 \leq i \leq 8$. The query Q_1 takes the union of the gene descriptions in SWISS-PROT and GENBANK. The `propagate default` clause specifies that annotations should be propagated in the default way, i.e., according to where data is copied from. Since the gene description `CC` is common to both SWISS-PROT and GENBANK, the corresponding addresses a_2 and a_8 of `CC` in SWISS-PROT and GENBANK respectively are merged together in the result of Q_1.

In general, an annotation is not restricted to be an address, it can also be a comment about a value such as its correctness and quality. As these comments are carried along query transformations, they give a sense of the overall quality of the resulting database that is generated by the query. DBNotes also allow the users to specify exactly how the annotations should propagate. This is done using the custom propagation scheme of pSQL. For example, the query Q_2 propagates the annotations associated with each ID value and Desc value of a tuple to the output Desc value according to the custom propagation scheme which is specified as "`propagate ID TO Desc, Desc TO Desc`".

Q_2:
select distinct Desc
from SWISSPROT
propagate ID TO Desc,
Desc to Desc

Result of Q_2:

Desc
CC $\{a_1, a_2\}$
ED $\{a_3, a_4\}$

Q_3:
select distinct s.Desc
from SWISSPROT s,
GENBANK g
where s.Desc = g.Desc
propagate default-all

Q_4:
select distinct g.Desc
from SWISSPROT s,
GENBANK g
where s.Desc = g.Desc
propagate default-all

Result of Q_3 or Q_4:

Desc
CC $\{a_2, a_8\}$

There is also a third propagation scheme supported by DBNotes, called the default-all propagation scheme. In this propagation scheme, annotations are propagated according to where data is copied from according to *all* equivalent queries. The reason for introducing the default-all scheme is because two equivalent SQL queries may not always generate the same annotations in the results under the default propagation scheme. On the other hand, the default-all propagation scheme gives the user an invariant semantics; two equivalent SQL queries under the default-all propagation scheme will always generate the same annotations in the results. The queries Q_3 and Q_4 above are equivalent but they generate different annotations in results under the default scheme (Q_3 returns the annotation a_2 while Q_4 returns the annotation a_8). However, they generate the same annotations in the results under the default-all propagation scheme. The result is shown above on the right. We refer the interested reader to [2] for more details about DBNotes. We note that recently, there is also interest in building a database system for managing and querying the accuracy and lineage of data [30].

5 Directions for Further Research

Motivated by the need to trace the provenance of data in scientific databases, we have identified some challenges below where we believe that research advances in these areas are important steps towards the overall goal of tracing the provenance of data in scientific databases.

Tracing Provenance through General Transformations Earlier works on tracing provenance have been largely database language and model specific. It happens often, however, that a database is created as a result of executing a program script or interpreting high-level abstractions given between two data sources or manual data entry. A complete system for tracing provenance should be able to handle these general transformations as well. At a minimum level, one should be able to treat a transformation step as a black-box and explain that a piece of data is generated by that transformation step using certain parameters. On the other end of the spectrum, one could also go at length to explain how a transformation step derived a piece of data. For data that has been manually entered, a complete description of provenance should involve the author, date and documentation on why that entry was made.

- **Computing provenance lazily** In this direction of research, a detailed analysis of a transformation step to compute the provenance of data is made only when the provenance is sought for. It would be interesting to see how earlier research ideas for tracing provenance in database model and language specific settings and research in *program slicing* [15] can be applied to the problem of computing the provenance of data in the result of a procedure or program execution.

- **Computing provenance eagerly** In this direction of research, the provenance of data is computed as data is being transformed. In this way, the provenance of any data is immediately available in the result when sought for. As discussed in Section 2.1, initial research along this direction has been proposed but further work on formalizing the semantics and properties of such systems is required.

- **Querying Provenance** Complementary to the problem of computing provenance is the problem of querying provenance. Examples of such queries include "Find all records generated by Alice at the Zurich Genetic Laboratory" or "Who else is looking at this record?". These queries are not always straightforward to answer since the provenance of each record entry may not have been stored in the derived database. We may also require extensions to existing query languages to issue such queries.

Semantic Archiving and Querying Many databases are often well-structured and organized and come with semantic constraints (e.g., schema, key or foreign key constraints). Most existing techniques for archiving or versioning data, however, fail to exploit these properties as described in Section 2.2. Classical diff techniques rely on computing the minimum edit distance between two versions and storing the smallest "delta" that would give rise to the minimum edit distance. The end result is an archive that is the most compact and gives good performance for retrieving historical versions. Unfortunately, the archive is not always amenable to reasoning about the changes that has occurred to each semantic object.

An ideal archiver should not only be able to give a compact representation of historical versions of data but also exploit the semantic constraints so that the continuity of each semantic object is either preserved in the archive or easily reconstructed. Recent work [4] is one step towards semantic archiving and needs to be extended to handle more general settings.

Complementary to semantic archiving is the ability to pose historical queries to the archive. Examples of such queries include "retrieve Alice's record in 1998", "report the changes that has occurred to Alice since 2000", or "retrieve the date when Alice moved from Seattle to Santa Cruz". There are some connections to research in temporal databases [28], and needs to be explored.

6 Concluding Remarks

We have motivated the data provenance problem for scientific databases and general database applications. We have also described some previous research work on data provenance from the database research community and briefly described the DBNotes prototype developed at UC Santa Cruz. Although there has been some existing work on data provenance, we believe that there are still many important problems that are not satisfactorily solved and deserve attention for further research.

References

[1] P. Bernstein and T. Bergstraesser. Meta-Data Support for Data Transformations Using Microsoft Repository. *IEEE Data Engineering Bulletin*, 22(1):9–14, 1999.

[2] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 900–911, Toronto, Canada, 2004.

[3] P. Buneman and I. Foster. Workshop on Data Derivation and Provenance. http://www-fp.mcs.anl.gov/˜foster/provenance/.

[4] P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving Scientific Data. *ACM Transactions on Database Systems (TODS)(SIGMOD/PODS Special Issue)*, 29(1):2–42, 2004.

[5] P. Buneman, S. Khanna, and W. Tan. Why and Where: A Characterization of Data Provenance. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 316–330, London, United Kingdom, 2001.

[6] P. Buneman, S. Khanna, and W. Tan. On Propagation of Deletions and Annotations Through Views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 150–158, Wisconsin, Madison, 2002.

[7] S. S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 26–37, Tucson, Arizona, 1997.

[8] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change Detection in Hierarchically Structured Information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 493–504, Montreal, Canada, 1996.

[9] S. Chien, V. Tsotras, and C. Zaniolo. Efficient Management of Multiversion Documents by Object Referencing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 291–300, Roma, Italy, 2001.

[10] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[11] Y. Cui and J. Widom. Lineage Tracing for General Data Warehouse Tranformations. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 471–480, Roma, Italy, 2001.

[12] Y. Cui, J. Widom, and J. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179–227, 2000.

[13] Concurrent Versions System: The open standard for version control. http://www.cvshome.org.

[14] DBCAT, The Public Catalog of Databases. http://www.infobiogen.fr/services/dbcat/, cited 5 June 2000.

[15] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *SIGPLAN Notices*, 39(4):229–243, 2004.

[16] R. Hull. E-services: a look behind the curtain. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 1–14, San Diego, California, 2003.

[17] T. Lee, S. Bressan, and S. Madnick. Source Attribution for Querying Against Semi-structured Documents. In *Workshop on Web Information and Data Management (WIDM)*, Washington, DC, 1998.

[18] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-Centric Management of Versions in an XML Warehouse. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 581–590, Roma, Italy, 2001.

[19] Merriam-Webster. Merriam-Webster Online - The Language Center.
http://www.m-w.com/home.htm.

[20] W. Miller and E. Myers. A file comparison program. *Software-Practice and Experience*, 15(11):1025–1040, 1985.

[21] H. U. A. Museums. Provenance Research.
http://www.artmuseums.harvard.edu/research/provenance/index.html.

[22] E. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[23] T. M. M. of Art. Provenance Research Project.
http://www.metmuseum.org/collections/department.asp?dep=22&full=1.

[24] M. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, 1(4):364–370, 1975.

[25] K. C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery (JACM)*, 26:422–433, 1979.

[26] W. Tan. Containment of relational queries with annotation propagation. In *Proceedings of the International Workshop on Database and Programming Languages (DBPL)*, Potsdam, Germany, 2003.

[27] W. Tichy. RCS - A system for Version Control. *Software – Practice and Experience*, 15(7):637–654, 1985.

[28] K. Torp, C. S. Jensen, and R. T. Snodgrass. Effective Timestamping in Databases. *The VLDB Journal*, 8(3-4):267–288, 2000.

[29] Y. R. Wang and S. E. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 519–538, Brisbane, Queensland, Australia, 1990.

[30] J. Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. Technical report, Stanford University, 2004. http://dbpubs.stanford.edu/pub/2004-40.

[31] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 91–102, Birmingham, United Kingdom, 1997.

[32] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245–1262, 1989.

[33] K. Zhang and D. Shasha. Fast algorithms for unit cost editing distance between trees. *Journal of Algorithms*, 11(6):581–621, 1990.