Provenance In Dynamic Data Systems*

Jing Zhang
University of Michigan

H.V. Jagadish
University of Michigan

Abstract

Most digital data sets are subject to modifications. For example, scientific data may be updated according to the new experimental results, and sales data updated periodically according to new sales made. We often have data derived from these digital data sets.

Our concern in this paper is the provenance of such derived data. Can we explain what a particular derived datum depends on, even if a value used in its derivation has since been modified. Can we determine if a particular derived value is still valid without performing full view maintenance. Questions of this sort are likely to arise when we derive results from modifiable data.

We present in this paper an overview of problems that arise in this context, with regard to fine-grain data provenance, and outline solutions to some of these problems.

1 Introduction

Digital data is ubiquitous. The ease of storing, querying, and manipulating it has led to an explosion of its use. We are consumers of digital data when we read blogs, check emails, watch videos, etc. Digital data[1], once created, can be used to derive more data. The derivations can serve many different purposes, e.g., reformatting, filtering out irrelevant information, aggregating a big volume of data into a small volume for better human comprehension, etc. We are all consumers of derived data when we check hotel ratings, visit price comparison sites, and even when we read blogs that rely on some data or statistics.

A consumer of a derived datum may request the provenance of the derived datum for one or more reasons, e.g., determining the reliability of the derived datum, debugging the derivation process, meeting the audit requirements, etc. The provenance of a derived datum includes both the derivation process and the specific source data used to derive this datum. This provenance can be computed and stored when the derived datum is generated [3, 6] or when the database operations in the derivation process take place [1], also known as eager approaches [13], or retrieved only when it is requested [4], also known as lazy approaches [13].

An eager approach usually couples the provenance computation with the derivation process and modifies the derivation process for this purpose: e.g., rewriting the derivation query [6, 7] or propagating annotations during the evaluation of the derivation query [8, 5]. The storage overhead of this approach soon becomes intractable if more and more derived data keep being generated. Thus, this approach is not practical in applications where derivations of data are frequent.

On the other hand, in an lazy approach, the provenance computation is decoupled from the derivation process and thus there is neither need for storage nor for modifications to the derivation process. However, this approach needs to handle modifications of the source data [14], i.e., insertion, deletion, updates, after the derivation. This is because the modifications after the derivation of a given derived datum may remove the provenance of the given datum or invalidate it.

Studying provenance in the presence of modifications is important since modifying existing data sets is a rather common practice, e.g., new experimental results being inserted, existing figures being corrected, erroneous facts being deleted and etc. A single modification to a data set can affect either part of the data set or the complete data set. Multiple modifications can be applied sequentially to the same part of a data set.

In this paper, we focus on retrieving provenance in the lazy way in the presence of modifications and on determining the effect of the modifications on selected derived data. Given a set of modifications that have taken place to the source data set after the derivation of a derived datum, we need to answer the following provenance related

*This work was supported in part by NIH grant #U54DA021519.

[1]In this paper, unless specified otherwise, we use the terminology "data" to refer to the concept "digital data".

questions.

1. What is the provenance of a given derived datum, even when (part of) the provenance is not in the current data set?
2. What is the part of the provenance of a given derived datum that is not affected by the modifications?
3. Is a given derived datum affected by the set of modifications?
4. How is the given derived datum affected?
5. What in the source data or derivation can explain the absence of an expected item in the derived data?

The above five provenance questions can be categorized into four groups. The first question is about the retrieval of the *lost* provenance due to the modifications, which will be discussed in Section 3. The second question is about the retrieval of the *not-affected* part of the provenance from the modifications, which will be discussed in Section 4. The third and the fourth questions are about figuring out the effect on some selected derived data by examining how their provenance has been affected by the modifications, which will be discussed in Section 5. The fifth question has been addressed in [2] and [9], with the former explaining what manipulations in the derivation process (which is a query plan or a workflow) lead to the absence of some expected derived data and the latter explaining what values in the source data leads to that, and both will be reviewed very briefly in Section 6.

2 Provenance In Dynamic Data Sets

We focus our discussion on relational databases, and consider four types of operations on a database: one is querying the database and the other three are modifying the database – inserting tuples, deleting tuples and updating tuples in place. During an interval of time, queries and modifications take place interleaved. We assume that those operations–queries and modifications–are linearized.

The problem we want to solve is to *answer the provenance questions shown in Section 1, given (1) a relational database at the current time point, (2) a log of operations over a period of time till the current time point, (3) a sequence of delta tables resulting from the modifications over that time period.* Depending on the specific provenance question at hand, not all of the given information is needed.

In order to answer the first provenance question, a baseline approach is to adopt temporal databases. With a temporal database, the contributing source tuples can be retrieved by querying the proper historical database using tracing queries [4], or by referencing to the historical database by tuple IDs of the contributing source tuples [5, 8]. However, there are a couple of downsides to this baseline approach. First, the existing databases need to be re-deployed using temporal databases if they want to provide provenance support. Second, the temporal databases have potentially inferior performance in comparison to the normal databases due to the need for a sophisticated timestamp mechanism [10]. Third, a log and an archive of overwritten values is required to enable any kind of retrieval of the historical information in the database. However, it is also well known that logs and archives are expensive to maintain and can be large in size. For example, most systems only keep logs spanning over a certain period of time backward, and the length of the time period depends on the specific application in question. Thus, it is desirable to keep the logs and archives as small as possible, which means only the very necessary information that is needed to retrieve lost provenance should be stored. Temporal databases, as designed for a different and arguably more general purpose than lost provenance retrieval, are not economical for lost provenance retrieval purpose, because there is unneeded information in logs/archives, such as the timestamp information.

In order to answer the second question, logs and delta tables are not needed. A technique that can decorrelate the nested subqueries and eliminate joins of the input tables will be sufficient. The decorrelation technique is similar, however a little different, to the one used in the complex query evaluation [12].

In order to answer the third and fourth provenance questions, a baseline approach can be adopting the incremental view maintenance [11]. However, this approach has a downside of unnecessary computation because the view maintenance updates the complete derived result set instead of the selected derived tuple(s).

In order to answer the fifth question, it is necessary to find what steps in the derivation have stopped the source tuples that would otherwise produce the expected result tuples from going through [2]. Moreover, to make the expected tuples to be in the result, some attribute values in the failing source tuples can be changed in order to make these source tuples go through the evaluation and produce the expected result tuples in the final result set [9].

In this paper, we are seeking for a solution to our problem such that

1. the solution can be applied with as few changes as possible to the existing databases, e.g., without re-deploying them;
2. the amount of computation in the solution should scale with the size of selected result tuples whose provenance is requested, and scale with the size of the provenance requested, e.g., if a user is only interested in part of the provenance of a result tuple, she should not pay for the cost of computing the un-

requested part of the provenance.

3 Retrieval Of Lost Source Provenance

To answer the first provenance question, we need to retrieve the provenance of a given derived tuple whether or not it is still in the current source database [14]. In order to enable this type of provenance retrieval over an existing database, we need to have three extra data structures added to the existing database, and then the retrieval can make use of them to find the provenance efficiently.

First, we define the provenance of a given derived tuple as follows. It is a modified version of the definition introduced in [4].

Definition 1. Given a database D of tables $T_1, ..., T_n$, a query Q and a derived tuple t, there exists a set of tables $T_1', ..., T_n'$ such that

- $T_i' \subseteq T_i$, where $i = 1, ..., n$
- $\{t\} = Q(T_1', ..., T_n')$
- $\forall T_k' : \forall t' \in T_k' : Q(T_1', ..., T_{k-1}', \{t'\}, T_{k+1}', ..., T_n') \neq \emptyset$

Notice that if a single table has more than one instance in the query, each instance is considered as a separate table.

Second, we describe the three extra data structures we need in order to retrieve the possibly overwritten provenance.

1. We need a log, denoted as *provenance log*, recording the operations that have taken place over a time period till the current time point. Every entry records one operation and each entry has a unique log ID, which can be used to identify the operation in this entry.
2. We associate with each tuple in the current database an extra attribute, denoted as *since*, storing a log ID, which indicates the operation that introduced this tuple into the database.
3. We also associate with each table in the current database a so-called *shadow table* that keeps the tuples that were once in the database table but have been removed at some time point. In particular, the shadow table has the same schema as the database table except for two extra attributes storing log IDs, denoted as *begin* and *end*, with *begin* indicating the operation that introduced the tuple into the database and *end* indicating the operation that removed the tuple from the database.

With these three data structures, we can have classical tracing queries modified to make use of them and retrieve the lost provenance. Details are in [14].

These three data structures incur certain space overhead. However, some of it can be avoided. First of all, the provenance log does not need to take extra space in practice, since all the database management systems keep some kind of logs and the provenance log can be implemented as a view over the system maintained logs. This is almost always possible since the really vital attributes in the provenance log are the log ID and the operation, which turn out to be the very basic information an average system log will keep. As for the space overhead due to the attribute *since*, the number of cells of this attribute is equal to the number of tuples in the database. Since the database tuples usually have multiple attributes and some of them are of more space-costly data types than integer type, the total cost of this extra attribute in integer type is only a fraction of the total size of the database. As for the shadow tables, they are most costly as compared to the other two data structures. The size of shadow tables grows with the number of tuples that have been updated or removed. Therefore, in a database with an moderate amount of changes of data, the space cost due to shadow tables is acceptable; and if the size of shadow tables grows above some threshold, special treatments can be adopted, e.g., compression of rarely used shadow tables or most ancient part of a shadow table. However, these treatments also mean that the queries over the shadow tables will take more time.

The book-keeping of these three data structures incurs some time overhead during the execution of an operation. In particular, when an operation takes place in the database, the provenance log should record it, the shadow tables might need to be updated. The former leads to a very small time cost, while the latter depends on how many tuples are updated or removed during this operation.

4 Retrieval Of Partial Provenance

The complete provenance can be overwhelmingly large for a data item in the derived result set of a complex derivation process. A good results explanation interface must guide a user through this large amount of provenance information. A typical user may explore only a small part of this provenance depending on the question they have in mind. For example, a user debugging the data errors in a database may only want the part of provenance within tables that are error-prone and not care about the part of provenance within tables that are known to be error-free.

In terms of a system architecture, a simple design of a usable system is to leave the provenance system itself untouched and to layer a user interface on top of it, to filter and present appropriate portions of the provenance. However, unless all provenance has been precomputed, this system architecture leads to unnecessary work: much provenance information is computed only to be thrown away by the user interface. If there have been modifications to source data, we may save ourselves the bother of invoking all the machinery of the previous section, if we can determine that it is going to be filtered out by the user interface.

This motivates the need to compute partial provenance as requested. We have developed techniques to compute the part of provenance within each source table separately, which is denoted as *atomic provenance*. Then any partial provenance can be computed as composing proper atomic provenance. The computation of the atomic provenance needs to employ some techniques to decorrelate the nested subqueries and to eliminate the joins of input tables. Meanwhile, the composition of atomic provenance involves matching the correlation values and the values of the joining attributes in each atomic provenance.

5 Validating Derived Data

When modifications take place to a database, a user may be concerned whether her derived data will become invalid or out of date because of the modifications. For example, a previously derived average may change after new source tuples join the database. Therefore, the user may want to validate her selected result tuples given the modifications that have taken place after she derived the selected result tuples. View maintenance can be used for this purpose, but this can be expensive. To minimize the cost of the validation, we should focus on the selected result tuples and avoid the computation of other result tuples in the same result set. Therefore, we explore the dependencies of the selected result tuples on both the present and *absent* source tuples; and use the discovered dependencies and the effect of modifications on these present/absent source tuples to determine the effect of the modifications on the selected result tuples.

6 Explanation Of The Absence Of Expected Result Data

When some result tuples that are expected to be in the result set do not show up as expected, we can infer that some steps in the derivation must have filtered out some of their input data that are supposed to produce the expected result tuples in the end. In [2], such input data are defined to be *unpicked* [2] and such manipulations are defined to be *picky* manipulations [2] for these unpicked data. [2] proposed both a top-down and a bottom-up approaches to search over the derivation process, e.g., a query plan, to find the picky manipulations. Moreover, [9] showed that proper changes can be made to some attribute values in the source data that have previously failed to produce the expected result tuples, such that these modified source data can now go through the query evaluation and produce the expected result tuples.

The bulk of provenance research has focused on data present in the result. [2, 9] initiate a stream of work on data absent in the result. All of the issues discussed above, relating to changes in the source data, apply equally to questions of absence in the result as to questions of presence in the result. We intend to conduct future research in this direction.

7 Conclusions

Most data is subject to update. When source data is updated upon which derived data depends, provenance can provide a framework to reason about the effects of the update. This argues for the importance of provenance study in the presence of data set modifications.

In this paper, we outlined five provenance related questions a user may want to ask in the presence of data set modifications. For each of these questions, we suggested a possible solution strategy. We make no claim that this list of five questions is exhaustive. We look forward to fruitful discussions at the workshop regarding these and possible other questions to consider for fine grain data provenance over dynamic data.

References

[1] Buneman, P., Chapman, A., and Cheney, J. Provenance management in curated databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2006), ACM, pp. 539–550.

[2] Chapman, A., and Jagadish, H. V. Why not? In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data* (New York, NY, USA, 2009), ACM, pp. 523–534.

[3] Chapman, A. P., Jagadish, H. V., and Ramanan, P. Efficient provenance storage. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2008), ACM, pp. 993–1006.

[4] Cui, Y., and Widom, J. Practical lineage tracing in data warehouses. In *In ICDE* (1999), pp. 367–378.

[5] Foster, J. N., Green, T. J., and Tannen, V. Annotated XML: Queries and provenance. In *PODS* (Vancouver, B.C., June 2008).

[6] Glavic, B., and Alonso, G. Perm: Processing provenance and data on the same data model through query rewriting. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on* (mar. 2009), pp. 174 –185.

[7] Glavic, B., and Alonso, G. Provenance for nested subqueries. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology* (New York, NY, USA, 2009), ACM, pp. 982–993.

[8] Green, T. J., Karvounarakis, G., and Tannen, V. Provenance semirings. In *PODS '07: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (New York, NY, USA, 2007), ACM, pp. 31–40.

[9] Huang, J., Chen, T., Doan, A., and Naughton, J. F. On the provenance of non-answers to queries over extracted data. *Proc. VLDB Endow. 1*, 1 (2008), 736–747.

[10] Jensen, C. S., and Lomet, D. B. Transaction timestamping in (temporal) databases. In *In Proceedings of the 27th VLDB Conference* (2001), pp. 441–450.

[11] Ramakrishnan, R., Ross, K. A., Srivastava, D., and Sudarshan, S. Efficient incremental evaluation of queries with aggregation. In *In SIGMOD* (1994), pp. 204–218.

[12] Seshadri, P., Pirahesh, H., and Leung, T. Y. C. Complex query decorrelation. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering* (Washington, DC, USA, 1996), IEEE Computer Society, pp. 450–458.

[13] Tan, W.-C. Research problems in data provenance. *IEEE Data Engineering Bulletin 27* (2004), 45–52.

[14] Zhang, J., and Jagadish, H. Lost source provenance. In *EDBT '10: Proceedings of the 13th International Conference on Extending Database Technology* (2010).