Provenance in Databases: Past, Current, and Future

Wang-Chiew Tan*
UC Santa Cruz
wctan@cs.ucsc.edu

Abstract

The need to understand and manage provenance arises in almost every scientific application. In many cases, information about provenance constitutes the proof of correctness of results that are generated by scientific applications. It also determines the quality and amount of trust one places on the results. For these reasons, the knowledge of provenance of a scientific result is typically regarded to be as important as the result itself. In this paper, we provide an overview of research in provenance in databases and discuss some future research directions. The content of this paper is largely based on the tutorial presented at SIGMOD 2007 [11].

1 Overview of Provenance

The word *provenance* is used synonymously with the word *lineage* in the database community. It is also sometimes referred to as *source attribution* or *source tagging*. Provenance means *origin* or *source*. It also means *the history of ownership of a valued object or work of art or literature* [26]. The knowledge of provenance is especially important for works of art, as it directly determines the value of the artwork. The same applies to digital artifacts or results that are generated by scientific applications. Information about provenance constitutes the proof of correctness of scientific results and in turn, determines the quality and amount of trust one places on the results. For these reasons, the provenance of a scientific result is typically regarded to be as important as the result itself. There are two granularities of provenance considered in literature: *workflow (or coarse-grained) provenance* and *data (or fine-grained) provenance*. In what follows, we provide an overview of workflow and data provenance. However, the focus of this paper is on data provenance, which is described in the rest of this paper (Sections 2 to 4).

Workflow (or coarse-grained) provenance: In the scientific domain, a workflow is typically used to perform complex data processing tasks. A *workflow* can be thought of as a program which is an interconnection of computation steps and human-machine interaction steps. *Workflow provenance* refers to the record of the entire history of the derivation of the final output of the workflow. The amount of information recorded for workflow provenance varies. It may include a complete record of the sequence of steps taken in a workflow to arrive at some dataset. In some cases, this entails a detailed record of the versions of softwares used, as well as the models and makes of hardware equipments used in the workflow. In addition to providing a proof of correctness

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

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

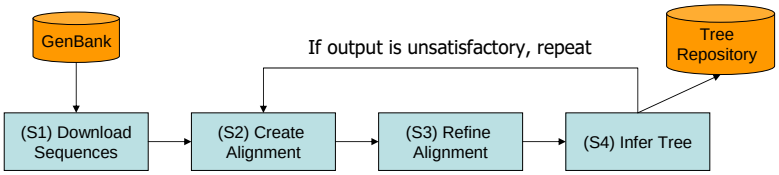


Figure 1: An example of a workflow from [16].

to the final workflow output, workflow provenance can also be useful for avoiding duplication of efforts; With appropriate bookkeeping of inputs taken by parts of the workflow, it is possible to identify parts of the workflow that need not be repeated across different execution runs.

Example 1: A simple example of a workflow from [16] is depicted in Figure 1. Arrows denote the flow of data, while boxes are used to indicate data processing steps. This workflow describes the steps taken to construct a phylogenetic tree from a set of DNA sequences. The workflow starts with step (S1) which downloads a set of DNA sequences from the GenBank repository. The second step (S2) takes the DNA sequences and runs an external sequence alignment program to generate a sequence alignment. Details of how a sequence alignment is constructed from multiple DNA sequences are "hidden" by the external program (i.e., the external program is a blackbox). Step S3 involves interaction with a biologist. The biologist examines the sequence alignment obtained from (S2) and may improve on the quality of the sequence alignment output by manually adjusting gaps inserted by the alignment program. The last step (S4) takes the edited alignment as input and produces a phylogenetic tree as output. There are in fact many steps involved in (S4) (see [16] for a detailed explanation). However, step (S4) abstracts the process of constructing a phylogenetic tree from the sequence alignment as another blackbox. The provenance of an execution of this workflow may include a record of the version of the GenBank repository used, the DNA sequences used, the software and version of the software used for sequence alignment, as well as the decisions made by the biologist in editing the alignment. □

As described in Example 1, an external process in step (S2) is involved in the workflow. In general, external processes do not possess good properties for a detailed analysis of the transformation since such details are typically hidden. Hence, the workflow provenance for this step is usually *coarse-grained*. That is, only the input, output and the software used by an external process are recorded.

Data (or fine-grained) provenance: In contrast, *data (or fine-grained) provenance* gives a relatively detailed account of the derivation of a piece of data that is in the result of a transformation step. A particular case of data provenance that is of interest to the database community and for which there have been considerable research efforts is when the transformation is specified by a database query. More precisely, suppose a transformation on a database D is specified by query Q, the provenance of a piece of data t in the output of $Q(D)$ is the answer to the following question: *Which parts of the source database D contribute to t according to Q?*

This is the subject of research of [18], where the authors described algorithms for computing data provenance in the relational framework. We give an example of data provenance in the relational setting next.

Source database D:

Employee

empid	dept
e977	CS
e132	EE
e657	BME

Department

deptid	budget
BME	1200K
CS	670K
EE	890K
MATH	230K

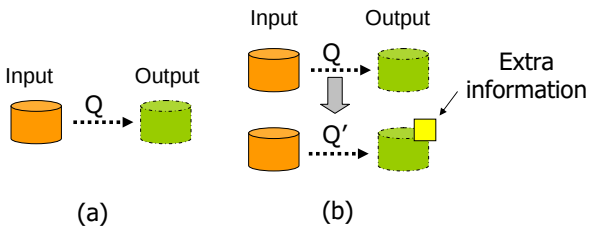
Query Q:

```
SELECT  e.empid, e.dept, d.budget
FROM    Employee e, Department d
WHERE   e.dept = d.deptid
```

Output of $Q(D)$:

empid	dept	budget
e657	BME	1200K
e977	CS	670K
e132	EE	890K

Figure 2: An example of a data transformation with SQL.



	'90	...	'97	...	'00	'01	'02	'03	...	'05	'06	'07
A	[31]	...		...			[10]	[30]	...	[4]	[17]	[7]
		...		...					...		[21]	[13]
		...		...					...		[22]	[20]
		...		...					...			[23]
		...		...					...			[29]
NA		...	[32]	...	[18]	[9]			...		[14]	

(c)

Figure 3: (a) Non-Annotation (*NA*) approach, (b) Annotation (*A*) approach, (c) a timeline for data provenance research efforts using either *NA* or *A* approach.

Example 2: Suppose D and Q are the database and query, respectively, shown in Figure 2. The result of executing Q against D is also shown on the right of the same figure. The source tuples (e657, BME) and (BME,1200K) contribute to the output tuple (e657,BME,1200K) according to Q. In particular, observe that some source tuples, such as (e132,EE), play no role in contributing to the output tuple (e657,BME,1200K) according to Q. The basic idea behind the algorithms proposed in [18] is to compute data provenance by analyzing the underlying algebraic structure of relational queries. Provenance is computed and aggregated according to the underlying algebraic operators used in query on an operator-by-operator basis. □

Data provenance is the focus of this paper and we shall elaborate more on data provenance in subsequent sections. Readers who are interested in workflow provenance may find the following references useful: A survey of provenance research related to scientific data processing and scientific workflow systems [5, 19] and a survey on provenance research in E-science [28].

All of the existing research efforts on data provenance adopt one of two contrasting approaches for computing data provenance: (1) *Non-annotation approach* vs. (2) *Annotation approach*. Techniques for computing data provenance that use the non-annotation approach allow the execution of a transformation function Q as it is. See Figure 3(a), which shows a normal execution of Q. That is, Q is executed on an input database to generate an output database. In order to compute the provenance of a piece of output data, it is typically the case that the input and output database, as well as the definition of Q, are analyzed to arrive at an answer. In contrast, techniques for computing provenance that use the annotation approach (see Figure 3(b)) carry additional information to the output database. In order to compute the extra information, the original transformation Q is usually modified to another transformation Q' so that when Q' is applied on the input database, it generates an output database that is identical to that generated by Q applied on the input database, as well as the additional information. With this approach, the provenance of a piece of output data can typically be derived by analyzing the extra information.

A timeline where the research efforts are classified according to the two approaches is tabulated in Figure 3(c). We shall discuss past research efforts (i.e., mainly research efforts prior to 2005) and current research efforts (i.e., mainly research efforts between 2005 and 2007) on data provenance in Sections 2 and 3 respectively. We conclude with some future research directions in Section 4.

2 Data Provenance: Past

As described in Section 1, the problem of computing data provenance in the relational framework was studied in [18]. It is easy to see that the need to compute data provenance applies not only to tuples of relations, but also to data that occur at different levels of granularities in a hierarchical or tree-like structure. This observation was made in [9], where the authors described a hierarchical data model and an associated query language for manipulating data in this hierarchical model. The provenance of a piece of data, at any granularity, in the result

of a monotone query can be obtained by analyzing the syntax of the query. In [9], the authors also made a distinction between *why* and *where-provenance*. The type of provenance studied by [18] is essentially why-provenance. Where-provenance, on the other hand, is a description of the locations in the input database which contain values where a piece of output data is copied from. To see the difference between why and where-provenance, consider Example 2 again. The why-provenance of the output tuple (e657,BME,1200K) according to Q consists of the two source tuples as described in Example 2. However, the where-provenance of the value "BME" in the output tuple (e657,BME,1200K) is the location pointed by the dept attribute in the source tuple (e657,BME). In other words, the "BME" value of the output tuple is copied from the "BME" value of the source tuple (e657,BME) and not from the "BME" value of the source tuple (BME,1200K). This is because the query Q extracts "BME" from e.dept (and not d.deptid). Observe that in contrast to the why-provenance of the output tuple (e657,BME,1200K), the where-provenance of "BME" of the same output tuple completely disregards the source Department relation.

Prior to [9] and [18], there has been a few similar research efforts [31, 32] targeted at resolving the data provenance problem. The authors of [32] proposed to build the functionality of computing data provenance into a database system using the non-annotation approach. Their motivation for using the non-annotation approach was to support provenance tracing in a database visualization environment, where large datasets are usually involved. It is therefore infeasible to associate additional information to every datum in these datasets for computing provenance. The main idea in [32] was to allow a user to register data processing functions and their corresponding inverse functions in a database system. When given a specific piece of output data to invert, an inversion planner module within the database system would infer which inverse function to apply and construct an execution plan by invoking the appropriate functions in the database system. However, since not all functions are invertible, a user is also allowed to register *weak inverses* instead. Intuitively, a weak inverse is an inverse function that approximates provenance; It may only return a subset of the desired provenance or more than what is required. A separate verification function is required to examine that the answers returned by the weak inverse are indeed answers to provenance. A fundamental drawback of this technique is that the user is required to provide (weak) inverse functions and their corresponding verification functions. Subsequent research efforts by [9] and, respectively, [18] that were described earlier, overcome this limitation by computing data provenance through analyzing the syntax and, respectively, algebraic structure of the queries.

The work of [31] first made the idea of using an annotation approach to compute provenance explicit. They proposed a polygen model ("poly" for "multiple" and "gen" for "source") that is able to track which originating data sources and intermediate data sources were used to generate an output data of interest. In [31], operational definitions on how one can compute the originating sources and intermediate sources of an attribute value over basic relational algebra operators were given.

The polygen idea was followed up by [10], where a similar set of operational definitions (called *propagation rules* in [10]) for basic relational operators were given[1]. In [10], however, the authors made clear that annotations (and not only originating sources) associated with source data can be propagated from source to output based on the propagation rules. Furthermore, the propagation rules were designed to propagate annotations based on where data is copied from (i.e., where-provenance). In particular, the relationships between *locations* of data in the input and output database were formalized through the propagation rules given in [10]. One of the problems studied in [10] is the *annotation placement problem*: *Given a query Q, source database D, a view $V = Q(D)$, and an annotation, denoted as $*$, placed in the view V, decide whether there is location to place the annotation $*$ in D so that $*$ propagates to the desired location in V and nowhere else.* If such a placement of $*$ in D exists, it is called a "side-effect-free annotation". The study of the annotation placement problem is important for understanding the bidirectional transport of annotations between the source database and the view. The authors showed a dichotomy in the complexity of the annotation placement problem for Select-Project-Join-Union (SPJU) queries: It is NP-hard to decide if there is a side-effect-free annotation for a project-join relational

[1] In [10], the authors had natural join instead of cartesian product in the set of basic relational algebra operators.

query even in the special case where the join is always performed before projection. On the other hand, there is a polynomial-time algorithm for deciding whether there is a side-effect-free annotation for SPJU queries which do not simultaneously contain both project and join operators. In fact, the annotation placement problem was later shown to be DP-hard in [30]. In [17], the authors showed that many of the complexity issues disappear for key-preserving operations, which are operations that retain the keys of the input relations.

3 Data Provenance: Current

In this section, we describe research efforts that mainly occur between 2005 and 2007, as shown in Figure 3(c). Our discussion will center around two research projects, DBNotes [4, 15] and SPIDER [1, 14], recently developed at UC Santa Cruz.

3.1 DBNotes

The work of DBNotes builds upon ideas developed in [10, 30]. DBNotes is an annotation management system for relational database systems. In DBNotes, every attribute value in a relation can be tagged with multiple annotations. When a query is executed against the database, annotations of relevant attribute values in the database are automatically propagated to attribute values in the result of the query execution. The queries supported by DBNotes for automatic annotation propagation belong to a fragment of SQL queries that corresponds roughly to select-project-join-union queries. In its *default* execution mode, annotations are propagated based on where data is copied from (i.e., where-provenance). As a consequence, if every attribute value in the database is annotated with its address, the provenance of data is propagated along, from input to output, as data is transformed by the query. An example of annotations propagated in the default manner is shown below:

Source database D:

Employee

empid	dept
e977 (a_1)	CS (a_2)
e132 (a_3)	EE (a_4)
e657 (a_5)	BME (a_6)

Department

deptid	budget
BME (b_1)	1200K (b_2)
CS (b_3)	670K (b_4)
EE (b_5)	890K (b_6)
MATH (b_7)	230K (b_8)

Query Q:

SELECT e.empid, e.dept, d.budget
FROM Employee e, Department d
WHERE e.dept = d.deptid
PROPAGATE default

Output of $Q(D)$:

empid	dept	budget
e657 (a_5)	BME (a_6)	1200K (b_2)
e977 (a_1)	CS (a_2)	670K (b_4)
e132 (a_3)	EE (a_4)	890K (b_6)

In this example, every attribute value in the source relations, Employee and Department, is annotated with a unique identifier. For instance, the attribute value 670K is annotated with the identifier b_4. The query Q has an additional "PROPAGATE default" clause, which means that we are using the default execution mode as explained earlier. By analyzing the annotations that are propagated to $Q(D)$, we can conclude that the value "BME" in $Q(D)$ was copied from "BME" in the Employee relation (and not the "BME" in Department relation). If the SELECT clause of Q had been "e.empid, d.deptid, d.budget" instead, then the annotation associated with "BME" in $Q(D)$ would be b_1 instead of a_6. Hence, equivalent queries may propagate annotations differently. This presents a serious difficulty as it means that the annotations (or provenance answers) that one obtains in the result is dependent on the query plan chosen by the database engine. DBNotes resolves this problem through a novel propagation scheme, called the *default-all* propagation scheme. In this scheme, all annotations of every equivalent formulation of a query are collected together. Consequently, propagated annotations are invariant across equivalent queries. This scheme can thus be viewed as the most general way of propagating annotations. In our example, the "BME" value in $Q(D)$ will consist of both annotations a_6 and b_1 under the default-all scheme. At first sight, the default-all scheme seems infeasible because the set of all equivalent queries is infinite in general. In [4], a practical and novel implementation that avoids the enumeration of every equivalent query is described. The key insight is that for monotone relational queries, all relevant annotations can in fact be determined by evaluating every query in a finite set of queries. Such a finite set can always be obtained, and is not too big in general. DBNotes also allows one to define *custom* propagation schemes. In this scheme, the user can specify where annotations should be retrieved from input relations. The custom scheme is especially useful when the user is, for example, only interested in retrieving annotations from a particular database, perhaps due

to its authority, over other databases. In [15], the query language of DBNotes was extended to allow querying of annotations. Techniques were also developed to explain the provenance and flow of data through query transformations via the analysis of annotations.

Extensions to DBNotes. In DBNotes, as well as [10], annotations can only be placed on a column of a tuple of a relation[2]. In other words, annotations can only be associated with attribute values only, and not tuples or relations. In [21], an extension is made so that annotations can be placed on any subset of attributes of a tuple in a relation. A *color algebra* that can query both values and annotations is also described. They showed that for unions of conjunctive queries, the color algebra is complete with respect to *color relational algebra queries*. A *color relational algebra query* is a query that when applied on an color database (i.e., relations with extra columns for storing annotations) returns another color database. They also showed that every operator in the color algebra is necessary for the completeness result to hold. In [20], a similar completeness result is proven for full relational algebra instead of unions of conjunctive queries; The color algebra of [20] is shown to be complete with respect to color relational algebra queries.

In [29], the idea of associating annotations with data is further extended to allow annotations to be placed on an arbitrary collection of data in a database. A query is used to capture the collection of data of interest and the query is then associated with the desired annotations in a separate table. Similarly, one can associate a collection of data with another collection of data by using two queries that capture the collections of data of interest respectively, and then associating the queries together in a separate table.

Expressivity of languages that propagate annotations. Since many languages that manipulate annotations (or provenance) were proposed, a natural question is the comparative expressive power of these query languages. For example, one natural question is the following: How does the propagation scheme for originating sources as proposed in [31] compare with the default propagation scheme of DBNotes? Is one more expressive than the other? The work of [7] addressed this question. They showed that the default propagation scheme of DBNotes is as expressive as the propagation scheme for originating sources proposed in [31]. To show this result, they defined a query language that manipulates annotations as "first-class citizens", and showed that the propagation schemes of [31] and DBNotes are equivalent in expressive power to a certain class of queries in their language.

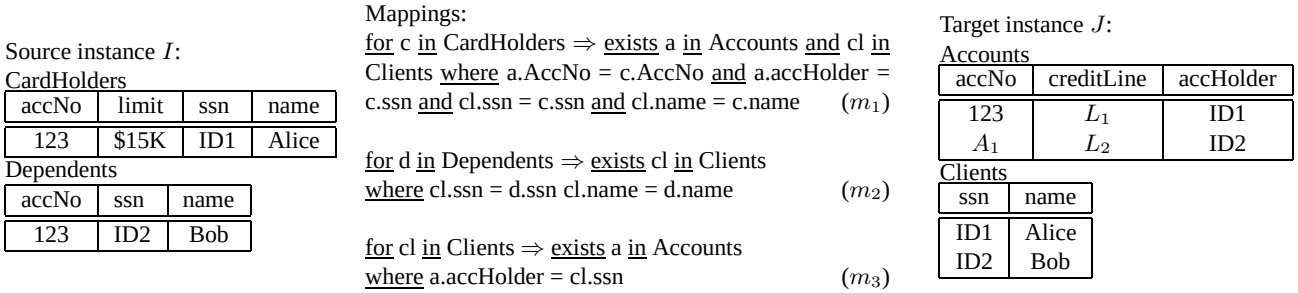
3.2 SPIDER

In this section, we describe a recent work on computing provenance over schema mappings that uses the non-annotation approach.

Schema mappings are logical assertions of the relationships between an instance of a source schema and an instance of the target schema. They are primary building blocks for the specification of data integration, data exchange and peer data management systems. A fundamental problem in integration systems is the design and specification of schema mappings, which typically takes a lot of time and effort to get it right [3, 24].

SPIDER [1, 14] is a system that facilitates the design of mappings by allowing mapping designers to understand, debug, and refine schema mappings at the level of mappings, through the use of examples. The idea behind SPIDER is very much like debuggers for programming languages which allow programmers to understand, debug, and refine their programs by running their programs on some test cases. The main approach that SPIDER uses to explain the semantics of mappings is through descriptions of the provenance (resp. flow) of data in the target instance (resp. source instance) through chains of possibly recursive mappings. These descriptions are called *routes* and intuitively, they describe how data in the source and target instances are related and constrained via mappings. In SPIDER, a mapping designer can either display routes ending at selected target data (i.e., trace the provenance of target data) or display routes starting at selected source data (i.e., trace the flow of source data). We describe an example of routes next.

[2]The same applies to [31], where originating sources are associated with attribute values.



The source schema consists of two relational schemas, CardHolders and Dependents, and the target schema consists of two relational schemas, Accounts and Clients. There are three mappings, m_1, m_2 and m_3, written in a query-like notation as shown in the middle of the figure above. Intuitively, the first mapping m_1 asserts that for every tuple in the CardHolders relation, there exists a tuple in the target Accounts relation and a tuple in Clients whose corresponding accNo, accHolder, ssn and name values are equal to the accNo, ssn, ssn, and name values, respectively, of the Cardholders tuple. The mapping m_2 asserts that every Dependents tuple has a corresponding Clients tuple whose ssn values coincide. The last mapping m_3 is a constraint on the target instance that says that the existence of a Clients tuple implies the existence of an Accounts tuple where the ssn value of the former is equal to the accHolder value of the latter tuple.

Given the schemas and the mappings, a mapping designer may wish to understand the mappings by executing them against a source instance I shown on the left of the figure above. A target instance J that conforms to the target schema and also satisfies the mappings is shown on the right. Such a target instance may be obtained by executing the schemas and mappings on a data exchange system such as Clio [25] or, by directly reasoning about the semantics of the mappings. In J, the values L_1, L_2 and A_1 represent possibly distinct unknown values for credit limits and account number. Since an account cannot be created without an account number, a mapping designer may probe on A_1 to understand how A_1 was formed in the exchange process. In response to the probe, SPIDER displays a route (shown below), starting from a source tuple in I that ultimately leads to the target tuple in J that contains the probed A_1.

$$\text{Dependents(123,ID2,Bob)} \xrightarrow{m_2} \text{Clients(ID2,Bob)} \xrightarrow{m_3} \text{Accounts}(A_2,L_2,\text{ID2})$$

Intuitively, the route explains that the Dependents tuple (i.e., Bob) in I leads to the Clients tuple (i.e., Bob) in J via mapping m_2, which in turn leads to the ID2 Accounts tuple in J via mapping m_3. Although not illustrated here, SPIDER also displays the bindings of variables in m_2 and m_3 that were used to derive each tuple in the route. By analyzing the route, a mapping designer may realize that the account number 123 in the Dependents tuple was somehow not copied over to the target and may hence refine or correct the mappings in the process.

The example above was kept simple for ease of exposition. In reality, mappings are usually not as simple as those shown in this example. They are usually larger and typically more complex. A major difficulty in computing routes is to reason about chains of possibly recursive mappings among schemas. Furthermore, the number of routes to illustrate to a mapping designer in response to a single probe may be overwhelming. In [14], an algorithm for computing routes that overcomes these difficulties has been developed. Their algorithm encodes the set of all routes, even when there may be exponentially many, in a compact polynomial-size representation. A demonstration of SPIDER is described in [1].

How-Provenance. Routes are different from why-provenance in that they not only describe which input tuples contribute to the existence of an output tuple, but also *how* the input tuples lead to the existence of the output tuple. Thus, compared to why-provenance, it is a more detailed explanation of the existence of an output tuple. In a recent paper [23], the authors described a method whereby provenance can be described using a *semiring of polynomials* in the context of datalog queries and introduced the term *how-provenance*. Semirings are similar to routes of SPIDER in that they capture the input tuples that contribute to an output tuple, as well as *how* they contribute to that output tuple. For example, let $R_1(A, B)$ be a binary relation with three tuples t_1, t_2 and t_3, where $t_1 = (1, 2)$, $t_2 = (1, 3)$ and $t_3 = (2, 3)$ and let $R_2(B, C)$ be another binary relation with three

tuples t_4, t_5 and t_6, where $t_4 = (2, 3)$, $t_5 = (3, 3)$ and $t_6 = (3, 4)$. The result of the query $\Pi_{A,C}(R_1 \bowtie R_2)$ consists of three tuples $(1, 3), (1, 4), (2, 3), (2, 4)$. The provenance polynomial for the output tuple $(1, 3)$ is $t_1t_4 + t_2t_5$, which describes that the output tuple $(1, 3)$ is witnessed by t_1 and t_4 or, t_2 and t_5. On the other hand, the why-provenance of $(1, 3)$ according to the query is simply the set of tuples $\{t_1, t_2, t_4, t_5\}$. Algorithms for calculating provenance semirings for datalog queries were described in [23]. In [22], an application of provenance semirings is described in collaborative data sharing: Updates that are propagated along peers carry along provenance semirings. These propagated semirings are subsequently utilized to trace the derivations of an update in order to determine whether an update should be filtered based on the trust conditions specified by participants of the data sharing system.

4 Data Provenance: Future

We have described some major research efforts in data provenance in the past two decades. In this section, we describe some possible future research directions.

Most research efforts on data provenance have focused on reasoning about the behavior of provenance and keeping track of annotations or metadata through SQL queries. While SQL queries are fundamental building blocks of many database applications, knowing how to reason about the provenance and flow of data through SQL queries alone is still insufficient for a complete end-to-end tracking of the provenance and flow of data in many database applications. For example, a Web application that is powered by a database backend may only use SQL queries to retrieve data from (or store data into) the underlying database system. Data that is retrieved may still undergo various transformations (e.g., cleansing or formatting transformations) before they are displayed on a Web page. To make matters worse, many Web applications today (e.g., mashups) are based on other Web applications where information is extracted and integrated though public application programming interfaces and appropriate programming languages. In particular, the process by which information is extracted and integrated is typically not described by SQL queries. Therefore, a major unsolved challenge for data provenance research is to provide a uniform and seamless framework for reasoning about the provenance (and flow) of data through different data transformation paradigms. We list three aspects of research on data provenance next that would make progress towards resolving this challenge.

Web applications and many other systems such as data warehouses, extract-transform-load systems behave very much like workflows, where data typically undergoes a sequence of transformations in different paradigms (e.g., SQL queries, C programs or Perl scripts.). Hence, one approach towards a solution for the above mentioned unsolved challenge is to examine whether one can combine the research efforts of workflow provenance and data provenance in a uniform manner. So far, the research efforts on workflow provenance and data provenance have been somewhat independent and disconnected. For the two threads of research to converge, extensions to the formalism for workflow provenance are needed so that nodes that represent external processes in a workflow need not be treated as a blackbox. In other words, whenever possible, one should be able to drill down and analyze the provenance of data generated by external programs, which are commonly used in workflows and typically abstracted as blackboxes by current techniques for computing workflow provenance. On the other front, techniques for computing data provenance need to be extended to handle constructs of more powerful languages (e.g., aggregates, iterators, and side-effects etc.). A recent promising research effort [12, 13] uses dependency analysis techniques, similar to program slicing and program analysis techniques from the programming language community, to analyze provenance over more complex database queries that includes relational queries with grouping and aggregates. Another approach towards a uniform framework for analyzing data provenance is to abstract different data transformation paradigms using higher-level declarative formalisms such as schema mappings. However, similar to the discussion earlier, mappings will need to be enriched to model constructs of more powerful languages such as aggregates, iterators, side-effects etc.

Another research direction that would make progress towards the unsolved challenge is to develop techniques for reasoning about or approximating the provenance of data that is generated by programs through the analysis

of the "blackbox behavior" of programs. In other words, even when the details of a program may not be available, one should still be able to derive the provenance of data generated by the program to a certain extent. Methods for resolving this challenge will be extremely useful in practice because in many cases, even when the details of an external program are available, they are typically too complex to be amenable to a systematic analysis.

The last research direction concerns archiving. This is a topic that bears close relationship to provenance and has not been discussed so far in this paper. Databases and schemas evolve over time. Necessarily, a complete record of provenance entails archiving all past states of the evolving database so that it becomes possible to trace the provenance of data to the correct version of the database or trace the flow of data in a version of the database that is not necessarily the most recent. Archiving is especially crucial for scientific data, where scientific breakthroughs are typically based on information obtained from a particular version of the database. Hence, all changes or all versions of the database must be fully documented for scientific results to remain verifiable. There have been some research efforts on archiving scientific datasets [6, 8]. However, two major challenges remain: (i) The first is to provide techniques for efficiently archiving versions of databases whose schema may also evolve over time. At the same time, the structure of the archive should still retain the semantics of data and relationships between entities across different versions of data as far as possible so that the archive can be meaningfully analyzed later. (ii) The second is to provide companion techniques to efficiently recover a version of the database from the archive obtained from (i), incrementally update the archive with a new version of data, as well as provide techniques to discover or analyze temporal-related properties in the archive and how entities evolve over time.

Recently, a number of applications of provenance have emerged in the context of probabilistic databases [2], schema mappings [14], and updates [22]. These applications require extensions to prior techniques for computing provenance. An interesting research direction would be to discover whether there are other applications of provenance that would require significant extensions to existing techniques or a completely new framework for computing provenance. For example, a recent workshop on provenance [27] suggests that security, information retrieval, dataflow or extract-transform-load scenarios etc. are some potential applications to investigate.

References

[1] B. Alexe, L. Chiticariu, and W.-C. Tan. SPIDER: a Schema mapPIng DEbuggeR. In *Very Large Data Bases (VLDB)*, pages 1179–1182, 2006. (Demonstration Track).

[2] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Very Large Data Bases (VLDB)*, pages 953–964, 2006.

[3] P. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1–12, 2007.

[4] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. *Very Large Data Bases (VLDB) Journal*, 14(4):373–396, 2005. A preliminary version of this paper appeared in the VLDB 2004 proceedings.

[5] R. Bose and J. Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Survey*, 37(1):1–28, 2005.

[6] P. Buneman, A. Chapman, and J. Cheney. Provenance Management in Curated Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 539–550, 2006.

[7] P. Buneman, J. Cheney, and S. VanSummeren. On the Expressiveness of Implicit Provenance in Query and Update Languages. In *International Conference on Database Theory (ICDT)*, pages 209–223, 2007.

[8] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving Scientific Data. *ACM Transactions on Database Systems (TODS)*, 29(1):2–42, 2004. A preliminary version of this paper appeared in the ACM SIGMOD 2002 proceedings.

[9] P. Buneman, S. Khanna, and W.-C. Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, pages 316–330, 2001.

[10] P. Buneman, S. Khanna, and W.-C. Tan. On Propagation of Deletions and Annotations Through Views. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems (PODS)*, pages 150–158, 2002.

[11] P. Buneman and W.-C. Tan. Provenance in Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1171–1173, 2007. (Tutorial Track).

[12] J. Cheney. Program Slicing and Data Provenance. *IEEE Data Bulletin Engineering*, December 2007.

[13] J. Cheney, A. Ahmed, and U. A. Acar. Provenance as Dependency Analysis. In *Database Programming Languages (DBPL)*, pages 138–152, 2007.

[14] L. Chiticariu and W.-C. Tan. Debugging Schema Mappings with Routes. In *Very Large Data Bases (VLDB)*, pages 79–90, 2006.

[15] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. DBNotes: A Post-It System for Relational Databases based on Provenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 942–944, 2005. (Demonstration Track).

[16] S. Cohen, S. Cohen-Boulakia, and S. B. Davidson. Towards a Model of Provenance and User Views in Scientific Workflows. In *Workshop on Data and Integration in Life Sciences (DILS)*, pages 264–279, 2006.

[17] G. Cong, W. Fan, and F. Geerts. Annotation Propagation for Key Preserving Views. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 632–641, 2006.

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

[19] S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludascher, T. McPhilips, S. Bowers, M. K. Anand, and J. Freire. Provenance in Scientific Workflow Systems. *IEEE Data Bulletin Engineering*, December 2007.

[20] F. Geerts and J. V. den Bussche. Relational Completeness of Query Languages for Annotated Databases. In *Database Programming Languages (DBPL)*, pages 127–137, 2007.

[21] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. In *International Conference on Data Engineering (ICDE)*, page 82, 2006.

[22] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update Exchange with Mappings and Provenance. In *Very Large Data Bases (VLDB)*, pages 675–686, 2007.

[23] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance Semirings. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems (PODS)*, pages 675–686, 2007.

[24] L. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. In *International Conference on Database Theory (ICDT)*, pages 28–43, 2007.

[25] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 805–810, 2005.

[26] Merriam-Webster OnLine. http://www.m-w.com.

[27] Workshop on Principles of Provenance (PrOPr), November 2007. http://homepages.inf.ed.ac.uk/jcheney/propr/.

[28] Y. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in E-Science. *SIGMOD Record*, 34:31–36, 2005.

[29] D. Srivastava and Y. Velegrakis. Intensional Associations Between Data and Metadata. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 401–412, 2007.

[30] W.-C. Tan. Containment of Relational Queries with Annotation Propagation. In *Database Programming Languages (DBPL)*, pages 37–53, 2003.

[31] Y. R. Wang and S. E. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *Very Large Data Bases (VLDB)*, pages 519–538, 1990.

[32] A. Woodruff and M. Stonebraker. Supporting Fine-grained Data Lineage in a Database Visualization Environment. In *International Conference on Data Engineering (ICDE)*, pages 91–102, 1997.