NOTE: THIS IS NOT THE ENTIRE PAPER, IT'S A SHORTENED VERSION PROVIDED BY SEBASTIAN DURING THE DATA PREPARATION COURSE AT UVA

Full text available at: http://dx.doi.org/10.1561/1900000006

Foundations and Trends® in
Databases
Vol. 1, No. 4 (2007) 379–474

DOI: 10.1561/1900000006

now
the essence of knowledge

Provenance in Databases: Why, How, and Where

James Cheney[1], Laura Chiticariu[2] and Wang-Chiew Tan[3]

[1] *University of Edinburgh, UK, jcheney@inf.ed.ac.uk*
[2] *IBM Almaden Research Center, San Jose, CA, USA, chiti@almaden.ibm.com*
[3] *University of California, Santa Cruz, CA, USA, wctan@cs.ucsc.edu*

Abstract

Different notions of provenance for database queries have been proposed and studied in the past few years. In this article, we detail three main notions of database provenance, some of their applications, and compare and contrast amongst them. Specifically, we review why, how, and where provenance, describe the relationships among these notions of provenance, and describe some of their applications in confidence computation, view maintenance and update, debugging, and annotation propagation.

Contents

Full text available at: http://dx.doi.org/10.1561/1900000006

1

Introduction

Provenance information describes the origins and the history of data in its life cycle. Such information (also called *lineage*) is important to many data management tasks. Historically, databases and other electronic information sources were trusted because they were under centralized control: it was assumed that trustworthy and knowledgeable people were responsible for the integrity of data in databases or repositories. As argued by Lynch [49], this assumption is no longer valid for online data. Today, data is often made available on the Internet with no centralized control over its integrity: data is constantly being created, copied, moved around, and combined indiscriminately. Because information sources (or different parts of a single large source) may vary widely in terms of quality, it is essential to provide provenance and other context information which can help end users judge whether query results are trustworthy.

Data warehouses [17] and curated databases [10] are typical examples where provenance information is essential. In both data warehouses and curated databases, tremendous (and often manual) effort is usually expended in the construction of the resulting database — in the former, in specifying the extract-transform-load (ETL) process and in the

latter, in incrementally adding and updating the database. In a sense, provenance adds value to the data by explaining how it was obtained. Hence, it is of utmost importance to understand the provenance of data in the resulting database, in order to check the correctness of an ETL specification or assess the quality and trustworthiness of curated data.

Provenance has been studied in several different areas of data management, such as scientific data processing [8, 29, 53] and database management systems [15, 57]. We focus on provenance for data residing in a database management system. A number of notions of provenance in databases have been proposed in the literature. The most common forms of database provenance describe relationships between data in the source and in the output, for example, by explaining *where* output data came from in the input [58, 13], showing inputs that explain *why* an output record was produced [27, 13] or describing in detail *how* an output record was produced [43]. Besides being interesting in their own right for understanding the behavior of queries, these forms of provenance have been used in the study of classical database problems, such as view update [14] and the expressiveness of update languages [11]. More recently, they have also been used in the study of annotation propagation [7, 11, 58] and updates across peer-to-peer systems [42].

In this article, we focus on these three existing notions of why-, how- and where-provenance in databases. We shall describe them, discuss their applications, and compare and contrast these different notions in the subsequent sections. In the rest of this introductory section, we provide a high-level overview of these different notions of provenance, and introduce notation that will be used throughout the rest of the article. Sections 2, 3 and 4 focus on why-, how- and where-provenance, respectively, including formal details and applications. Section 5 discusses the relationships among the approaches, including proofs or disproofs of some "folklore" properties which have been stated in the literature but not (to our knowledge) carefully formalized and proved. Finally, Section 6 concludes with a brief discussion of additional related work and research challenges.

We emphasize that there are numerous other notions of provenance that are not described in this article. For example, provenance is also an active topic of research in scientific workflow management system

Full text available at: http://dx.doi.org/10.1561/1900000006

community and in the file and storage systems community. This article focuses on provenance within databases, and we refer the interested reader to the surveys [8, 53], and a recent tutorial [29] for a discussion on provenance research in general, as well as in the workflow community. Recent workshops [33, 35] also provide insight into the different views of provenance by diverse research communities.

Even in database settings, there is work that does not fit neatly into the why-, where- and how-provenance framework we focus on here, including early work such as Wang and Madnick's Polygen model [58] and Woodruff and Stonebraker's work on lineage [61], as well as Cui et al.'s lineage model [27] and more recent work on the Trio system [6]. We have chosen to focus on the why-, where-, and how-provenance framework because there is now enough related research on these models area to justify a critical review and comparison. We have recast lineage and a simplification of the Trio model as instances of our framework, but the Polygen, Woodruff–Stonebraker, and full Trio models seem to resist this categorization. Our classification should therefore be viewed as a preliminary attempt towards a full understanding of provenance in databases. We return to this issue in Section 6.

1.1 Why, How and Where: An Overview

1.1.1 Why-Provenance

ALSO REFERRED TO AS LINEAGE ONLY

IT'S ABOUT DEFINING THE TUPLES IN THE STARTING TABLES THAT ALLOWED THE OUTPUT TUPLE TO BE GENERATED

Cui et al. [27] were among the first to formalize a notion of provenance, of data in the context of relational databases, called *lineage*. They associated each tuple t present in the output of a query with a set of tuples present in the input, called the *lineage* of t. Intuitively, the lineage of t is meant to collect all of the input data that "contributed to" t or helped to "produce" t. To illustrate, we use a simple example database of an online travel portal shown in Figure 1.1, where the labels $t_1, \ldots, t_8$ are used to identify the tuples. Consider the query Q_1[1] shown below, which asks for all travel agencies that offer external boat tours and their corresponding phone numbers by joining Agencies with ExternalTours

[1] Throughout the paper, we use SQL, relational algebra, and Datalog notation interchangeably, as convenient.

ALSO REFERRED TO AS WITNESSES

WHY PROVENANCE ACTUALLY FOCUSES ON THE # WITNESS BASIS AND NOT ALL THE WITNESSES

WITNESS BASIS ARE A COMPACT REPRESENTATION OF ALL WITNESSES

(FOR EACH TUPLE THERE MAY BE SEVERAL WITNESSES)

Agencies

	name	based_in	phone
t_1:	BayTours	San Francisco	415-1200
t_2:	HarborCruz	Santa Cruz	831-3000

ExternalTours

	name	destination	type	price
t_3:	BayTours	San Francisco	cable car	$50
t_4:	BayTours	Santa Cruz	bus	$100
t_5:	BayTours	Santa Cruz	boat	$250
t_6:	BayTours	Monterey	boat	$400
t_7:	HarborCruz	Monterey	boat	$200
t_8:	HarborCruz	Carmel	train	$90

Fig. 1.1 Our example database: an online travel portal.

on the name attribute, selecting tours by boat, and projecting on the name and phone attributes:

Q_1:
SELECT a.name, a.phone
FROM Agencies a, ExternalTours e
WHERE a.name = e.name AND
e.type='boat'

Result of Q_1:

name	phone
BayTours	415-1200
HarborCruz	831-3000

The result of Q_1 executed on our example database in Figure 1.1 is shown above on the right. According to Cui et al., the lineage of the output tuple (HarborCruz, 831-3000) is {Agencies(t_2), ExternalTours(t_7)}, where Agencies(t_2) and ExternalTours(t_7) denote the subinstances of Agencies and ExternalTours consisting of tuples t_2 and t_7, respectively. Intuitively, the two source tuples witness the existence of the tuple of interest, (HarborCruz, 831-3000), according to Q_1. Furthermore, each of the two source tuples justify the existence of the HarborCruz tuple. In other words, the source tuples t_2 and t_7 form a "proof" or "witness" for the HarborCruz output tuple according to Q_1, and no other source tuples are part of the witness since they do not contribute to the HarborCruz output tuple. Technically speaking, by "witness" we mean a subset of the input database records that is sufficient to ensure that a given output tuple appears in the result of a query.

As another example, the lineage of the output tuple (BayTours, 415-1200) is the union of the lineage of the intermediate

tuples — (BayTours, San Francisco, 415-2000, Santa Cruz, boat, $100) and (BayTours, San Francisco, 415-2000, Monterey, boat, $250) — before the projection operator is applied on name and phone. The union of the lineage of these two intermediate tuples gives {Agencies(t_1), ExternalTours(t_5,t_6)}. Observe that this lineage representation is not as precise as one may like as it does not specify that t_5 and t_6 do not need to coexist together in order to witness the BayTours output tuple. Indeed, $\{t_1, t_5\}$ and respectively, $\{t_1, t_6\}$ are two different witnesses for the BayTours tuple. This illustrates that not every tuple in the lineage is "necessary" for the output (BayTours, 415-1200) to be produced.

This intuition was formalized by Buneman et al. [13] who introduced the notion of *why-provenance* that captures the different witnesses. Their work is in the context of a semi-structured data model with a query language that is appropriate for that data model, but we shall restrict our discussion to the relational model with select–project–join queries here.

Like lineage, why-provenance is based on the idea of providing information about the witnesses to a query. Recall that a witness is a subset of the database records that is sufficient to ensure that a given record is in the output. There may be a large number of such witnesses because many records are "irrelevant" to the presence of an output record of interest. In fact, the number of witnesses can easily be exponential in the size of the input database. To avoid this problem, the definition of why-provenance restricts attention to a smaller number of witnesses. According to [13], the why-provenance of an output tuple t in the result of a query Q applied to a database D is defined as the *witness basis* of t according to Q. The witness basis is a particular set of witnesses which can be calculated efficiently from Q and D. This is generally much smaller than the full witness set. However, every witness contains an element of the witness basis, so the witness basis can be viewed as a compact representation of the set of all witnesses.

Going back to our example, the why-provenance of (BayTours, 415-1200) in the result of Q_1 is the set $\{\{t_1, t_5\}, \{t_1, t_6\}\}$. There are two witnesses, corresponding to $\{t_1, t_5\}$ and $\{t_1, t_6\}$, respectively. Intuitively, this tells us that the output tuple is witnessed by source tuples in two different ways according to Q_1: the first uses the tuples t_1 and

Instance I:

R

	A	B
t:	1	2
t':	1	3
t'':	4	2

Two equivalent queries:

$Q : Ans(x,y) :- R(x,y).$

$Q' : Ans(x,y) :- R(x,y), R(x,z).$

Output of $Q(I)$, $Q'(I)$:

A	B
1	2
1	3
4	2

Fig. 1.2 Example queries, input and output.

Instance I:

R

	A	B
t:	1	2
t':	1	3
t'':	4	2

Output of $Q(I)$

A	B	*why*
1	2	$\{\{t\}\}$
1	3	$\{\{t'\}\}$
4	2	$\{\{t''\}\}$

Output of $Q'(I)$

A	B	*why*
1	2	$\{\{t\},\{t,t'\}\}$
1	3	$\{\{t'\},\{t,t'\}\}$
4	2	$\{\{t''\}\}$

Fig. 1.3 Example showing that why-provenance is sensitive to query rewriting.

t_5, while the second uses the tuples t_1 and t_6. Observe that $\{t_1, t_5, t_6\}$ is not a minimal witness, since the query Q_1 requires witnesses to consist of exactly one tuple from Agencies, and one tuple from ExternalTours according to the `FROM` clause of Q_1.

The preceding discussion suggests that the witness basis may be tied to the structure of the query and it is therefore sensitive to how a query is formulated. To illustrate, consider the instance I and two equivalent queries Q and Q' shown in Figure 1.2. For conciseness, we use the Datalog conjunctive query notation to express Q and Q' here and throughout the paper as convenient. Consider the output tuple $(1,2)$ in the result of Q (and Q') applied to I shown in Figure 1.3. The witness basis of this output tuple is $\{\{t\}\}$, according to Q and I. However, even though Q' is equivalent to Q, the witness basis of the output tuple $(1,2)$ according to Q' and I is $\{\{t\},\{t,t'\}\}$.

Although equivalent queries may have different witness bases, Buneman et al. [13] showed that a subset of the witness basis, called the *minimal witness basis*, is invariant under equivalent queries. The minimal witness basis consists of all the *minimal* witnesses in the witness basis, where a witness is minimal if none of its proper subinstances is also a witness in the witness basis. For example, $\{t\}$ is a minimal witness for the output tuple $(1,2)$ in Figure 1.2. However, $\{t, t'\}$ is not a

minimal witness since $\{t\}$ is a subinstance of it and it is a witness to (1,2). Hence, the minimal witness basis is $\{\{t\}\}$ for this example. In a subsequent work by [14], minimal witnesses were used in the study of variants of the view deletion problem, which is that of finding source tuples to remove in order to delete a tuple from the view for select-project–join–union queries.

1.1.2 How-Provenance

Why-provenance describes the source tuples that witness the existence of an output tuple in the result of the query. However, it leaves out some information about *how* an output tuple is derived according to the query. To illustrate, consider the query Q_2 of Figure 1.4 which asks for all cities where tours are offered (assuming all agencies offer tours in the city they are headquartered). The result of Q_2 on the example database in Figure 1.1 is shown in the right of Figure 1.4. (Ignore the additional tags on the output tuples for now.) For the output tuple (San Francisco, 415-1200) in the result of Q_2, its why-provenance is $\{\{t_1\}, \{t_1,t_3\}\}$. This description tells us that t_1 alone, and t_1 with t_3 are each sufficient to witness the existence of the output tuple according to Q_2. However, it does not tell us about the structure of the proof that t_1 (as well as t_1 and t_3) help witness the output tuple according to Q_2. Although arguably obvious from the description of the query Q_2, the why-provenance does not tell us that the source tuple t_1 contributes twice to the output tuple: (1) t_1 contributes to the intermediary result of the inner query, and (2) it combines with that intermediary result to witness the output tuple. This intuition is formalized in [43] using

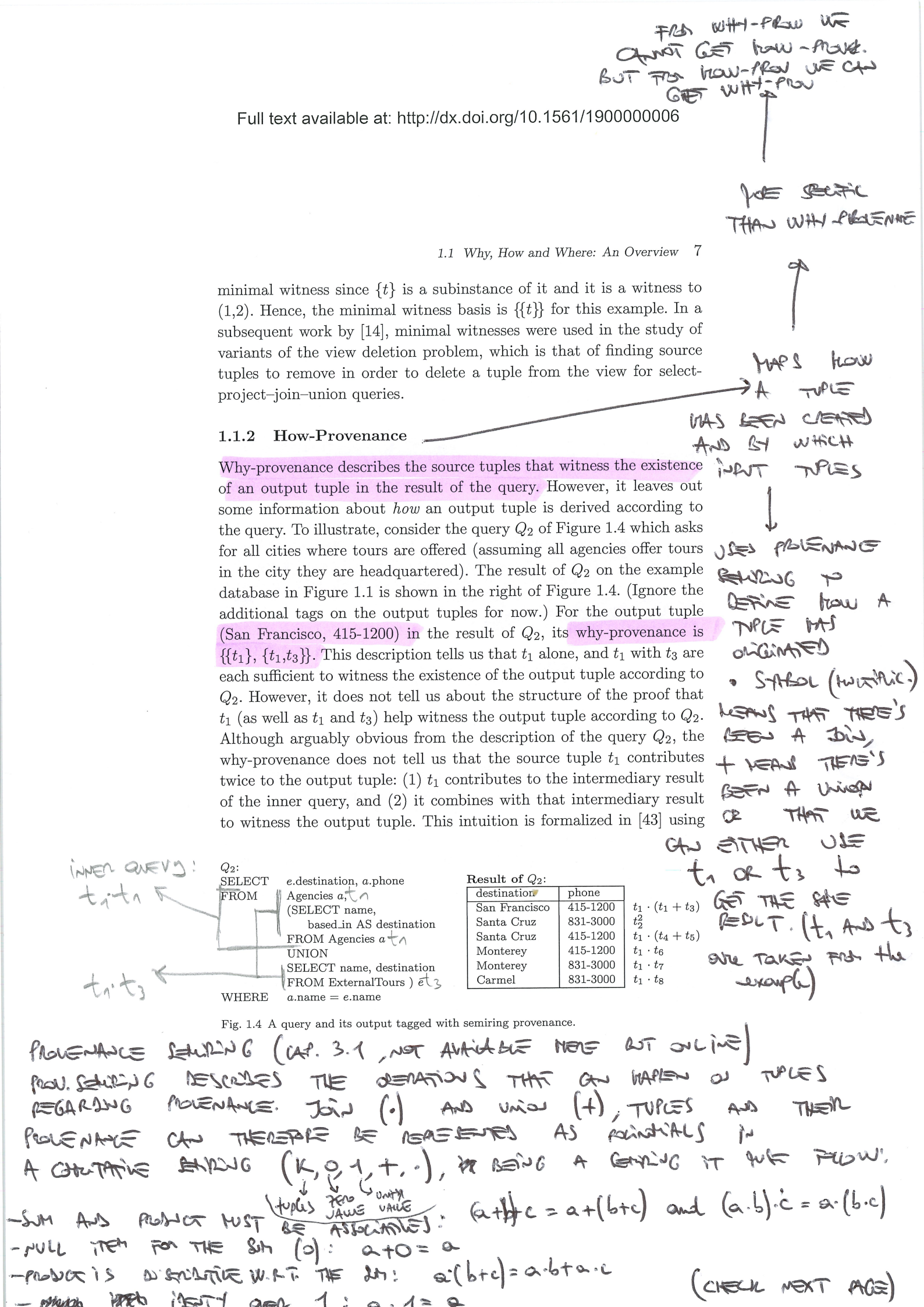
Q_2:

```
SELECT   e.destination, a.phone
FROM     Agencies a,
         (SELECT name,
             based_in AS destination
          FROM Agencies a
          UNION
          SELECT name, destination
          FROM ExternalTours ) e
WHERE    a.name = e.name
```

Result of Q_2:

destination	phone	
San Francisco	415-1200	$t_1 \cdot (t_1 + t_3)$
Santa Cruz	831-3000	t_2^2
Santa Cruz	415-1200	$t_1 \cdot (t_4 + t_5)$
Monterey	415-1200	$t_1 \cdot t_6$
Monterey	831-3000	$t_1 \cdot t_7$
Carmel	831-3000	$t_1 \cdot t_8$

Fig. 1.4 A query and its output tagged with semiring provenance.



PROVENANCE SEMIRING (CAP. 3.1, NOT AVAILABLE HERE BUT ONLINE)
PROV. SEMIRING DESCRIBES THE OPERATIONS THAT CAN HAPPEN ON TUPLES REGARDING PROVENANCE. JOIN ($\cdot$) AND UNION ($+$), TUPLES AND THEIR PROVENANCE CAN THEREFORE BE REPRESENTED AS POLYNOMIALS IN A COMMUTATIVE SEMIRING $(K, 0, 1, +, \cdot)$, K BEING A SEMIRING IT HAS TO BE:
(tuples, zero value, unity value)
- SUM AND PRODUCT MUST BE ASSOCIATIVE: $(a+b)+c = a+(b+c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- NULL ITEM FOR THE SUM (0): $a+0=a$
- PRODUCT IS DISTRIBUTIVE W.R.T. THE SUM: $a \cdot (b+c) = a \cdot b + a \cdot c$
- UNITY ITEM OVER 1: $a \cdot 1 = a$

(CHECK NEXT PAGE)

Instance I:
R

	A	B
t:	1	2
t':	1	3
t'':	4	2

Output of $Q(I)$

A	B	*how*
1	2	t
1	3	t'
4	2	t''

Output of $Q'(I)$

A	B	*how*
1	2	$t^2 + t \cdot t'$
1	3	$(t')^2 + t \cdot t'$
4	2	$(t'')^2$

Fig. 1.5 Example showing that how-provenance is sensitive to query rewriting.

provenance semirings. Intuitively, the provenance of the output tuple (San Francisco, 415-1200) is represented as a polynomial, which for this example is $t_1^2 + t_1 \times t_3$. The polynomials for each output tuple are shown on the right of the result of Q_2. The polynomial hints at the structure of the proofs by which the output tuple is derived. In this example, the polynomial describes that the output tuple is witnessed in two distinct ways: once using t_1 twice, and the other using t_1 and t_3. As we shall show, one can derive the why-provenance of an output tuple from its how-provenance polynomial. However, this example shows that the converse is not always possible.

It is easy to see that how-provenance is also sensitive to query formulations, since how-provenance is more general than why-provenance. Going back to our example queries shown on the top of Figure 1.2, Figure 1.5 illustrates that the how-provenance of the tuple $(1,2)$ in the output of $Q(I)$ is t according to Q, and respectively, $t^2 + t \times t'$ according to Q'.

Green et al. [43] formalize a notion of how-provenance for relational algebra in terms of an appropriate "provenance semiring", and extend their approach to handle recursive datalog. Subsequently, an interesting application of how-provenance appears in the context of ORCHESTRA [42, 44], a collaborative data sharing system in a network of peers interconnected through schema mappings. An extension of the semiring model of Green et al. [43] to schema mappings is used in ORCHESTRA to efficiently support trust-based filtering of updates, and incremental maintenance of peers' databases with updates in the system.

Earlier, Chiticariu and Tan proposed a notion of provenance over schema mappings called *routes* [21], and used it as a basis for SPIDER, a system for debugging schema mappings [3]. Given a schema mapping that relates a source and a target schema, routes describe *how* data in

the source instance is related to data in the target instance through the schema mapping. Hence, in retrospect, routes can be classified as a form of how-provenance over schema mappings.

1.1.3 Where-Provenance

Why-provenance describes all combinations of source tuples that witness the existence of an output tuple in the result of a query. In turn, how-provenance describes how the source tuples witness the output tuple. Buneman et al. also introduced a different notion of provenance, called *where-provenance* [13]. Intuitively, where-provenance describes where a piece of data is copied from. While why-provenance is about the relationship between source and output tuples, where-provenance describes the relationship between source and output *locations*. In the relational setting, a location is simply a column of a tuple in a relation, which precisely refers to a "cell" in a relation. The where-provenance of a value that resides in some location l in $Q(D)$ consists of *locations* of D from which the value in l was copied according to Q. Naturally, this requires that all the values that reside in the source locations of the where-provenance of l are equal to the value that resides at l. For example, the where-provenance of the value "HarborCruz" in the second output tuple in the result of Q_1 is the location (Agencies, t_2, name) (or simply, (t_2, name)) in our example database, since "HarborCruz" was copied from the name attribute of the tuple t_2 in the Agencies relation, according to Q_1.

Where-provenance is also not invariant under equivalent queries. To illustrate, consider the queries Q_1 (repeated from earlier) and Q_1'. The only difference between Q_1 and Q_1' is in the select clause. The first attribute of the select clause of Q_1 is a.name, whereas the first attribute of the select clause of Q_1' is e.name.

Q_1:
```
SELECT  a.name, a.phone
FROM    Agencies a, ExternalTours e
WHERE   a.name = e.name
        AND e.type='boat'
```

Q_1':
```
SELECT  e.name, a.phone
FROM    Agencies a, ExternalTours e
WHERE   a.name = e.name
        AND e.type='boat'
```

Clearly, the queries Q_1 and Q_1' are equivalent in that they produce the same resulting tuples for any given input database. However, the where-provenance of the output value "HarborCruz" is different under the two queries. As explained earlier, the where-provenance of "HarborCruz" in the output according to Q_1 is the location (t_2, name), since "HarborCruz" is copied from the name attribute of the tuple t_2 in Agencies. With Q_1', however, the where-provenance of "HarborCruz" is (t_7, name), since "HarborCruz" is copied from the name attribute of t_7 in ExternalTours. Arguably, the where-provenance of "HarborCruz" according to Q_1 and Q_1' is identical once we take the equality "a.name = e.name" into consideration in Q_1. However, as we shall discuss later with DBNotes, where-provenance is still not invariant under equivalent queries even after such "equality checks" are incorporated.

According to Buneman et al.'s definition of why and where-provenance [13], if a value v of a location of an output tuple t is not constructed by a query Q, then it must have been copied from values that reside in some source locations of a witness of t according to Q. As a consequence, the where-provenance of v consists of locations that can be found in tuples of the why-provenance of t. (We prove this more carefully in Section 5, Proposition 5.11.) For example, consider Q_1' that was described earlier and the output tuple (BayTours, 415-1200), which we denote as t. The why-provenance of t according to Q_1' is $\{\{t_1, t_5\}, \{t_1, t_6\}\}$. The where-provenance of the value "BayTours" at location (t, name) consists of two locations (t_5, name) and (t_6, name). Indeed, these two locations are among the locations of tuples in the witnesses of t. Observe that although t_1 is part of every witness for t according to Q_1', the locations of t_1 are not among the locations in the where-provenance of (t, name). On the other hand, the where-provenance of (t_1, phone) is (t, phone). But in general, it is possible for the why-provenance to contain tuples whose locations contribute nothing to the where-provenance of any part of the output.

One interesting application of where-provenance has been in the study of annotation–propagation and update languages [7, 14, 58]. Annotation–propagation is closely related to provenance: a given notion of provenance can be viewed as a method for propagating annotations

from the input to the output, whereas a given annotation–propagation semantics can be viewed as a form of provenance by placing distinct annotations on each part of the input and observing where they end up in the output.

The idea of forwarding provenance information during query execution was first explored in the Polygen system [58]. In Polygen, operational rules for forwarding information about the source databases, as well as the intermediate databases that contributed to the creation of an output piece of data, are defined for basic relational operators. The where-provenance propagation rules are similar to the rules that propagate *origin source tags* (i.e., references to original sources) in Polygen [58].

In Buneman et al.'s work on annotation propagation [14], as implemented in DBNotes [7], the where-provenance of a location in the result of a query determines the set of all annotations in the source database to be associated with that output location. This approach assumes queries involve select, project, join, and union only; the where-provenance of an output location is described through a set of *propagation rules*, one for each relational operator (i.e., select, project, join, union). In contrast to Polygen, these approaches propagate arbitrary annotations through a query, and not only information about source and intermediary databases.

DBNotes [7, 22] is an annotation management system for relational databases which builds upon previous ideas in where-provenance and Polygen. DBNotes propagates annotations from source locations to output locations based on where-provenance. Queries in DBNotes are also select–project–join–union queries except that they are expressed in declarative SQL-like expressions, and not relational operators as in [14]. Like [14], every location can be associated with zero or more annotations and these annotations are propagated to the output when a query is executed. The *default* annotation–propagation behavior of queries in DBNotes forwards annotations to an output location based on the where-provenance of that output location. For a simple example, consider the annotated relation I_a shown in Figure 1.6 and the queries Q and Q' from Figure 1.2. Each of the six locations of I_a is associated with one annotation, denoted as a_i, where $1 \leq i \leq 6$. The execution of

Annotated instance I^a:
R

	A	B
t:	1^{a_1}	2^{a_2}
t':	1^{a_3}	3^{a_4}
t'':	4^{a_5}	2^{a_6}

Output of $Q(I^a)$ (DEFAULT propagation):

A	B
1^{a_1}	2^{a_2}
1^{a_3}	3^{a_4}
4^{a_5}	2^{a_6}

Output of $Q'(I^a)$ (DEFAULT propagation):

A	B
$1^{a_1,a_3}$	2^{a_2}
$1^{a_1,a_3}$	3^{a_4}
4^{a_5}	2^{a_6}

Output of $Q(I^a)$, $Q'(I^a)$ (DEFAULT-ALL propagation):

A	B
$1^{a_1,a_3}$	$2^{a_2,a_6}$
$1^{a_1,a_3}$	3^{a_4}
4^{a_5}	$2^{a_2,a_6}$

Fig. 1.6 Example showing that where-provenance is sensitive to query rewriting.

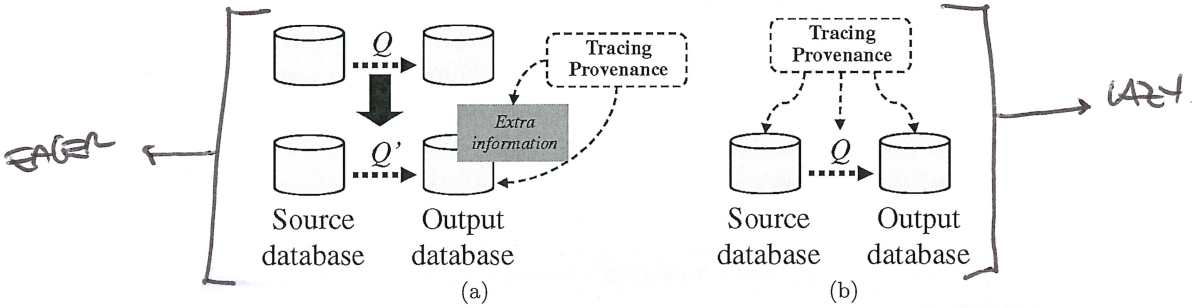
Q and respectively, Q' on I_a under the default propagation scheme produces the two annotated instances shown in Figure 1.6. In the output of Q, the annotation a_1 propagates from the value "1" of the source tuple t to the output value "1" of (1, 2) in $Q(I_a)$. This is because the value "1" of (1, 2) in $Q(I_a)$ is copied from the value "1" of t according to Q. In the case of Q', however, the value "1" of (1, 2) in $Q'(I_a)$ is copied from "1" of t or "1" of t' in I_a. Hence, two annotations, a_1 and a_3, appear with the value "1" of (1, 2) in $Q'(I_a)$. This simple example illustrates once more that where-provenance is sensitive under equivalent query formulations: while Q and Q' are equivalent, they produce different annotated results. In fact, the query Q'': $Ans(x,y) :- R(x,y)$, $R(z,y)$ is also equivalent to Q and it propagates both a_2 and a_6 to the values "2" in the output, whereas the two copies of value "1" in the output is annotated with a_1 and respectively, a_3.

If a query Q propagates annotations under the *default-all* propagation scheme in DBNotes, then equivalent formulations of Q are guaranteed to produce identical annotated results. In the default-all scheme, annotations are propagated based on where data is copied from according to *all* equivalent queries of Q. Hence, this propagation scheme can be perceived as a "better" method for propagating annotations for Q. The result of executing Q (or Q' or Q'') on I_a under the default-all scheme is shown in Figure 1.6. Observe that all annotations relevant for an output value are associated under the same output value in the default-all behavior, regardless of how the query is formulated. For this example, both "1"s in the default-all output are associated with a_1 and a_3. This is because Q', which is an equivalent query of Q, associates both annotations with the value "1". Similarly, both "2"s in the default-all output are associated with a_2 and a_6. This is because Q'' associates

both annotations with the value "2". In fact, given Q, DBNotes generates a finite set of equivalent queries (in this case, $\{Q, Q', Q''\}$) that captures all the relevant annotations that would be propagated by any equivalent query of Q.

1.2 Approaches in Computing Provenance: Eager vs Lazy

Along with our discussion of the three notions of provenance, we shall also give an overview of a few recent systems where provenance is an integral component, and describe the algorithms for computing provenance implemented in these systems. Figure 1.7(c) illustrates a classification of the systems we shall discuss in some detail in this paper, based on the approach each takes in computing provenance. There are two approaches for computing provenance: the *eager approach* and the *lazy approach*. In this article, we describe the basic ideas behind the two approaches, and defer the discussion of system implementation details to Sections 2–4.



	Eager Approach	*Lazy Approach*
Why-provenance		WHIPS (i.e., lineage) [25, 27]
How-provenance	ORCHESTRA [42, 44] Trio [2, 5, 52]	SPIDER [3, 21]
Where-provenance	DBNotes [7, 22]	

(c)

Fig. 1.7 Approaches in computing data provenance: (a) the eager approach; (b) the lazy approach. (c) A classification of recent systems for computing data provenance.

Figures 1.7(a) and 1.7(b) illustrate the two possible approaches for computing provenance. In the *eager approach* (also known as the *bookkeeping* or *annotation* approach), the query is re-engineered so that extra annotations are carried over to the output database during the transformation, to help answer provenance. As a consequence, the provenance of a piece of output data can usually be derived by examining the output database and the extra information. In the *lazy approach* (also known as *non-annotation* approach), provenance is computed when needed — by examining the source data, the output data, and the transformation. In contrast to the eager approach, the lazy approach does not require the re-engineering of the transformation for the purpose of carrying additional information to the output database.

Both approaches have advantages, as well as disadvantages, and they are appropriate in different scenarios. Since additional information is carried over and stored along with actual data in the output database, an eager approach involves a performance overhead during the execution of the transformation, as well as a space overhead for storing the extra information in the output. Recently, various schemes aimed at reducing the amount of extra information stored have been investigated in [31, 38, 54], and the problem of compressing or approximating provenance has been explored in [9, 16, 50]. However, the eager approach has the advantage that if the right additional information is propagated, provenance may be derived directly from the output database and the extra information, without examining the source database. Hence, an eager approach is useful in scenarios where the source data may become unavailable after the transformation. The lazy approach does not require the re-engineering of the transformation. Hence, it has the advantage that it can be readily deployed on an existing system without changes to the system, and furthermore, it does not incur any performance or storage overhead during the execution of the transformation. Thus, a lazy approach is useful when storage space is an issue, or, when it is not possible to modify the implementation of the query execution system. A disadvantage of the lazy approach is that deriving provenance usually involves sophisticated techniques for reasoning about the source database, the output database, and the

transformation. Hence, the lazy approach cannot be used if the source data becomes unavailable.

Although most existing work takes a distinctively eager or lazy approach, it might be interesting to consider hybrid approaches that take advantage of the best characteristics of both the approaches. In fact, the WHIPS lineage-tracking system [25, 27] combines eager and lazy ideas in its handling of queries involving negation and aggregation.

1.3 Notational Preliminaries

In this section, we introduce (largely standard) notation we shall use in the rest of the paper.

Let $\mathbf{D}$ be a finite domain of data values $\{d_1,\ldots,d_n\}$ and $\mathcal{U}$ a collection of *field names* (or attribute names). We will use the symbols U,V for (finite) subsets of $\mathcal{U}$. A record (or tuple) $t,t_1,t_2,t',u,\ldots$ is a function $U \to \mathbf{D}$, written as $(A_1{:}d_1,\ldots,A_n{:}d_n)$. A tuple assigning values to each field name in U is called U-tuple; e.g. $(A_1{:}d_1,\ldots,A_n{:}d_n)$ is a $\{A_1,\ldots,A_n\}$-tuple. We write *Tuple* for the set of all tuples and *U-Tuple* for the set of all U-tuples. We write $t{\bullet}A$ for the value of the A-field of t, $t[U]$ for the restriction of tuple t over $V \supseteq U$ to field names in U, and $t[A \mapsto B]$ for the result of renaming field A to B in t (assuming B is not already present in t). We sometimes write $(A{:}e(A))_{A\in U}$ to define a tuple $t{:}U$ such that $t{\bullet}A = e(A)$ for each $A \in U$.[2] Here, $e(A)$ is an expression parameterized by an unknown field name A. For example, if $t{:}V$ then we can express the projection $t[U]$ using this notation as $(A{:}t{\bullet}A)_{A\in U}$.

A *relation* or table $r{:}U$ is a finite set of tuples over U. Let $\mathcal{R}$ be a finite collection of *relation names*. A *schema* $\mathbf{R}$ is a mapping $(R_1{:}U_1,\ldots,R_n{:}U_n)$ from $\mathcal{R}$ to finite subsets of $\mathcal{U}$. A *database (or instance)* $I{:}(R_1{:}U_1,\ldots,R_n{:}U_n)$ is a function mapping each $R_i{:}U_i \in \mathbf{R}$ to a relation r_i over U_i.

We also define *tuple locations* as tuples tagged with relation names, written (R,t). We write $\mathit{TupleLoc} = \mathcal{R} \times \mathit{Tuple}$ for the set of all tagged tuples. We can view a database instance I equivalently as a finite set

[2] For readers familiar with lambda-calculus notation for function definition, note that $(A{:}e(A))_{A\in U}$ is equivalent to $\lambda A \in U \cdot e(A)$.

$\{(R,t) \mid t \in I(R)\} \subseteq \mathit{TupleLoc}$ of such tagged tuples according to a standard translation. We will also sometimes consider *field locations* that refer to a particular field of a tagged tuple. Formally, such a location is just a triple $(R,t,A) \in \mathcal{R} \times \mathit{Tuple} \times U$. We write $\mathit{FieldLoc}$ for the set of all locations.

We will use the following notation for (monotone) relational algebra queries:

$$Q ::= R \mid \{t\} \mid \sigma_\theta(Q) \mid \pi_U(Q) \mid Q_1 \bowtie Q_2 \mid Q_1 \cup Q_2 \mid \rho_{A \mapsto B}(Q)$$

Here, $\{t\}$ is a *singleton constant* $\{t\}$. Selections σ_θ filter a relation by retaining tuples satisfying some predicate θ. We leave the form of predicates unspecified (but typically include field equality tests $A = B$ and $A = d$). Projections $\pi_U(Q)$ replace each tuple t in a relation with $t[U]$, discarding any other fields. Join (or natural join) and union are standard; renaming is written $\rho_{A \mapsto B}(Q)$.

The precise semantics $Q(I)$ of a query Q evaluated against an instance I is described below. We review this standard definition only because we will be considering a number of variations on it later.

$$\begin{aligned}
(\{t\})(I) &= \{t\} \\
R(I) &= I(R) \\
(\sigma_\theta(Q))(I) &= \{t \in Q(I) \mid \theta(t)\} \\
(\pi_U(Q))(I) &= \{t[U] \mid t \in Q(I)\} \\
(Q_1 \bowtie Q_2)(I) &= \{t \mid t[U_1] \in Q_1(I), \quad t[U_2] \in Q_2(I)\} \\
(Q_1 \cup Q_2)(I) &= Q_1(I) \cup Q_2(I) \\
(\rho_{A \mapsto B}(Q))(I) &= \{t[A \mapsto B] \mid t \in Q(I)\}
\end{aligned}$$

Here, we assume that Q has the set of attributes V, denoted as $Q{:}V$, that $U \subseteq V$ in the case of projection, and that $Q_1{:}U_1, Q_2{:}U_2$ in the case of join.

As mentioned earlier, when convenient we also employ Datalog notation using nameless tuples, and assume familiarity with the standard translation between SPJRU queries and unions of conjunctive Datalog queries. For example, the query $\{(A(x,y) :- R(x,y), S(x,z)), (A(x,x) :- R(x,x))\}$ is equivalent to $(R \bowtie S) \cup \sigma_{A=B}(R)$, where we assume schema $R(A,B)$ and $S(A,C)$.

We also employ the following convention regarding partial functions (which is standard in, for example, programming language semantics). Formally, we can view a partial function $f: X \to Y$ as a total function $f: X \to Y \cup \{\bot\}$, where $\bot$ is a special, fresh constant not already present in Y, called "undefined". We write $Y_\bot$ to abbreviate $Y \cup \{\bot\}$, and we define $\mathrm{dom}(f) = \{x \in X \mid f(x) \neq \bot\}$.

One advantage of this convention is that it permits unambiguous definitions of operations with different behavior regarding undefinedness. For example, we will later make use of *strict* and *lazy* union operations. Strict union $\cup_S$ is defined as the union of two sets if both are defined, and undefined otherwise (that is, $X \cup_S \bot = \bot$), whereas lazy union $\cup_L$ differs from strict union in that it is undefined only if both sets are undefined (that is, $X \cup_L \bot = X$). We will define these operations more carefully later.

The various provenance semantics we shall consider will be defined by interpreting the language of relational queries over other classes of structures besides relations. A familiar example of this technique is interpreting queries over bags (multisets) instead of set relations. The semiring provenance semantics discussed earlier directly generalizes both relation and multiset semantics, and several other provenance semantics are instances of the semiring semantics.

Full text available at: http://dx.doi.org/10.1561/1900000006

References

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases.* Addison Wesley Publishing Co, 1995.

[2] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 1151–1154, 2006. (Demonstration Track).

[3] B. Alexe, L. Chiticariu, and W.-C. Tan, "Spider: A schema mapping debugger," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 1179–1182, 2006. (Demonstration Track).

[4] F. Bancilhon and N. Spyratos, "Update semantics of relational views," *ACM Transactions on Database Systems (TODS)*, vol. 6, pp. 557–575, 1981.

[5] O. Benjelloun, A. D. Sarma, A. Halevy, M. Theobald, and J. Widom, "Databases with uncertainty and lineage," *The VLDB Journal*, vol. 17, pp. 243–264, 2008.

[6] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom, "Uldbs: Databases with uncertainty and lineage," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 953–964, 2006.

[7] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, "An annotation management system for relational databases," *The VLDB Journal*, vol. 14, pp. 373–396, 2005. (A preliminary version of this paper appeared in the VLDB 2004 proceedings).

[8] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: A survey," *ACM Computing Survey*, vol. 37, pp. 1–28, 2005.

[9] 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)*, pp. 539–550, 2006.

[10] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, "Curated databases," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 1–12, 2008.

[11] P. Buneman, J. Cheney, and S. Vansummeren, "On the expressiveness of implicit provenance in query and update languages," *ACM Transactions Database Systems*, vol. 33, pp. 1–47, 2008.

[12] P. Buneman and D. Suciu (eds.), "Special issue on data provenance," *IEEE Data Engineering Bulletin*, vol. 30, 2007.

[13] P. Buneman, S. Khanna, and W.-C. Tan, "Why and where: A characterization of data provenance," in *Proceedings of the International Conference on Database Theory (ICDT)*, pp. 316–330, 2001.

[14] P. Buneman, S. Khanna, and W.-C. Tan, "On propagation of deletions and annotations through views," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 150–158, 2002.

[15] P. Buneman and W.-C. Tan, "Provenance in databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 1171–1173, 2007.

[16] A. P. Chapman, H. V. Jagadish, and P. Ramanan, "Efficient provenance storage," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 993–1006, 2008.

[17] S. Chaudhuri and U. Dayal, "Data warehousing and olap for decision support," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 507–508, 1997.

[18] J. Cheney, "Program slicing and data provenance," *IEEE Data Engineering Bulletin*, pp. 22–28, December 2007.

[19] J. Cheney, "Provenance, XML and the Scientific Web," in *ACM SIGPLAN Workshop on Programming Language Technology and XML (PLAN-X 2009)*, 2009. Invited paper.

[20] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance as dependency analysis," in *Proceedings of the International Workshop on Database and Programming Languages (DBPL)*, pp. 138–152, 2007.

[21] L. Chiticariu and W.-C. Tan, "Debugging schema mappings with routes," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 79–90, 2006.

[22] 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)*, pp. 942–944, 2005. (Demonstration Track).

[23] G. Cong, W. Fan, and F. Geerts, "Annotation propagation revisited for key preserving views," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 632–641, 2006.

[24] S. S. Cosmadakis and C. H. Papadimitriou, "Updates of relational views," *Journal of the Association for Computing Machinery (JACM)*, vol. 31, pp. 742–760, 1984.

[25] Y. Cui and J. Widom, "Lineage tracing in a data warehousing system," in *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 683–684, 2000. (Demonstration Track).

[26] Y. Cui and J. Widom, "Run-time translation of view tuple deletions using data lineage," Technical Report, Stanford University, 2001.

[27] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," *ACM Transactions on Database Systems (TODS)*, vol. 25, pp. 179–227, 2000.

[28] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, vol. 14, pp. 523–544, 2007.

[29] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 1345–1350, 2008.

[30] U. Dayal and P. A. Bernstein, "On the correct translation of update operations on relational views," *ACM Transactions on Database Systems (TODS)*, vol. 7, pp. 381–416, 1982.

[31] M. Y. Eltabakh, W. G. Aref, A. K. Elmagarmid, M. Ouzzani, and Y. N. Silva, "Supporting annotations on relations," in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pp. 379–390, 2009.

[32] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, "Data exchange: Semantics and query answering," *Theoretical Computer Science (TCS)*, vol. 336, pp. 89–124, 2005.

[33] First Workshop on Theory and Practice of Provenance (TaPP). http://www.usenix.org/event/tapp09, February 2009.

[34] J. N. Foster, T. J. Green, and V. Tannen, "Annotated XML: Queries and provenance," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 271–280, 2008.

[35] J. Freire, D. Koop, and L. Moreau, eds., *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop (IPAW 2008), Revised Selected Papers*, number 5272 in LNCS. Springer, 2008.

[36] F. Geerts and J. V. den Bussche, "Relational completeness of query languages for annotated databases," in *Proceedings of the International Workshop on Database and Programming Languages (DBPL)*, pp. 127–137, 2007.

[37] F. Geerts, A. Kementsietsidis, and D. Milano, "imondrian: A visual tool to annotate and query scientific databases," in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pp. 1168–1171, 2006. (Demonstration Track).

[38] F. Geerts, A. Kementsietsidis, and D. Milano, "Mondrian: Annotating and querying databases through colors and blocks," in *Proceedings of the International Conference on Data Engineering (ICDE)*, p. 82, 2006.

[39] F. Geerts and A. Poggi, "On BP-complete query languages on k-relations," in *Workshop on Logic in Databases (LID)*, 2008. Available online: http://conferenze.dei.polimi.it/lid2008/LID08geerts.pdf.

[40] T. J. Green, Personal communication, August 2008.

[41] T. J. Green, "Containment of conjunctive queries on annotated relations," in *Proceedings of the International Conference on Database Theory (ICDT)*, pp. 296–309, 2009.

[42] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen, "Update exchange with mappings and provenance," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 675–686, 2007.

[43] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 675–686, 2007.

[44] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen, "Orchestra: Facilitating collaborative data sharing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 1131–1133, 2007. (Demonstration Track).

[45] A. Halevy, M. Franklin, and D. Maier, "Principles of dataspace systems," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 1–9, 2006.

[46] A. M. Keller, "Choosing a view update translator by dialog at view definition time," in *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 467–474, 1986.

[47] P. G. Kolaitis, "Schema mappings, data exchange and metadata management," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 61–75, 2005.

[48] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pp. 233–246, 2002.

[49] C. A. Lynch, "When documents deceive: Trust and provenance as new factors for information retrieval in a tangled web," *Journal of the American Society for Information Science and Technology*, vol. 52, pp. 12–17, 2001.

[50] C. Ré and D. Suciu, "Approximate lineage for probabilistic databases," *Proceedings of the VLDB Endowment*, vol. 1, pp. 797–808, 2008.

[51] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting lineage for confidence computation in uncertain and probabilistic databases," Technical Report 2007-15, Stanford InfoLab, March 2007.

[52] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting lineage for confidence computation in uncertain and probabilistic databases," in *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 1023–1032, 2008.

[53] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Record*, vol. 34, pp. 31–36, 2005.

[54] D. Srivastava and Y. Velegrakis, "Intensional associations between data and metadata," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 401–412, 2007.

[55] W.-C. Tan, "Data annotations, provenance and archiving," PhD thesis, University of Pennsylvania, 2002.

[56] W.-C. Tan, "Containment of relational queries with annotation propagation," in *Proceedings of the International Workshop on Database and Programming Languages (DBPL)*, pp. 37–53, 2003.

[57] W.-C. Tan, "Provenance in databases: Past, current and future," *IEEE Data Engineering Bulletin*, vol. 30, pp. 3–12, 2007.

[58] 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)*, pp. 519–538, 1990.

[59] J. Widom, "Trio: A system for integrated management of data, accuracy and lineage," in *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pp. 262–276, 2005.

[60] J. L. Wiener, H. Gupta, W. Labio, Y. Zhuge, and H. Garcia-Molina, "The whips prototype for data warehouse creation and maintenance," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 557–559, 1997. (Demonstration Track).

[61] 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)*, pp. 91–102, 1997.